



Microchip TCP/IP Stack Help

Table of Contents

1 TCPIP Library	16
1.1 Introduction	17
1.1.1 Getting Help	17
1.2 Legal Information	18
1.3 Release Notes	19
1.4 Utilities	121
1.4.1 MPFS2	121
1.4.1.1 Building MPFS2 Images	121
1.4.1.2 Uploading Pre-built MPFS2 Images	122
1.4.1.3 Advanced MPFS2 Settings	122
1.4.1.4 MPFS2 Command Line Options	123
1.4.2 Microchip TCP/IP Discoverer	123
1.5 Getting Started	125
1.5.1 Peripheral Usage	125
1.5.2 Demo Board Information	125
1.5.2.1 PIC24FJ256GB110 Plug-In-Module (PIM)	125
1.5.2.2 PIC18F87J11 Plug-In-Module (PIM)	126
1.5.2.3 PIC32MX795F512L Plug-In-Module (PIM)	126
1.5.2.4 MRF24WB/MRF24WG PICtail Daughter Board	127
1.5.3 Wi-Fi Demo Board Hardware Setup	127
1.5.3.1 Explorer 16	128
1.5.3.2 PIC18 Explorer Development Board	128
1.5.3.3 Wi-Fi G Demo Board	129
1.5.4 Programming and First Run	129
1.5.5 Configure your WiFi Access Point	130
1.5.6 Connecting to the Network	131
1.5.7 Uploading Web Pages	131
1.5.8 Accessing the Demo Application	132
1.5.9 Configuring WiFi Security	132
1.6 Demo Information	134
1.6.1 Demo Compatibility Table	134
1.6.2 Available Demos	135
1.6.2.1 TCPIP WiFi G Demo	135
1.6.2.2 TCPIP WiFi Console Demo App	135
1.6.2.2.1 Standalone Commands	135
1.6.2.2.1.1 iwconfig Commands	136

1.6.2.2.1.2 ifconfig Commands	138
1.6.2.2.1.3 iwpriv Commands	139
1.6.2.2.1.4 iperf Example	140
1.6.2.3 TCPIP WiFi Demo App	141
1.6.2.3.1 TCPIP WiFi Demo Modules	141
1.6.2.3.1.1 Web Page Demos	141
1.6.2.3.1.2 E-mail (SMTP) Demo	145
1.6.2.3.1.3 Generic TCP Client	146
1.6.2.3.1.4 Generic TCP Server	147
1.6.2.3.1.5 Ping (ICMP) Demo	147
1.6.2.3.1.6 UART-to-TCP Bridge	148
1.6.2.3.1.7 Zero Configuration (ZeroConf)	149
1.6.2.4 TCPIP WiFi EasyConfig Demo	150
1.7 Using the Stack	153
1.7.1 Stack Architecture	153
1.7.2 How the Stack Works	153
1.7.2.1 Required Files	153
1.7.2.2 APP_CONFIG Structure	154
1.7.2.3 Main File	154
1.7.2.3.1 Initialization	154
1.7.2.3.2 Main Loop	155
1.7.2.4 Cooperative Multitasking	155
1.8 Configuring the Stack	158
1.8.1 Hardware Configuration	158
1.8.1.1 External Storage	158
1.8.1.2 PIC32MX7XX Config	159
1.8.1.3 PIC18F87J11 Config	160
1.8.2 Address	160
1.8.2.1 MAC Address	160
1.8.2.2 IP Address	161
1.8.3 Protocol Configuration	162
1.8.3.1 Protocol Macros and Files	163
1.8.3.2 Additional Features	164
1.8.3.3 Sockets	165
1.8.3.3.1 Memory Allocation	165
1.8.3.3.2 Socket Types	166
1.8.3.3.3 Initialization Structure	167
1.8.3.3.4 UDP Sockets	167
1.8.3.3.5 BSD Sockets	168
1.9 Library Interface	169
1.9.1 Stack API	169

1.9.1.1 Announce	169
1.9.1.1.1 Stack Members	169
1.9.1.1.1.1 AnnounceIP Function	169
1.9.1.1.1.2 DiscoveryTask Function	169
1.9.1.2 ARCFOUR	170
1.9.1.2.1 Public Members	170
1.9.1.2.1.1 ARCFOUR_CTX Structure	170
1.9.1.3 ARP	170
1.9.1.3.1 Public Members	171
1.9.1.3.1.1 ARPRegisterCallbacks Function	171
1.9.1.3.1.2 arp_app_callbacks Structure	171
1.9.1.3.1.3 ARP_REQ Macro	172
1.9.1.3.1.4 ARP_RESP Macro	172
1.9.1.3.2 Stack Members	172
1.9.1.3.2.1 ARPInit Function	172
1.9.1.3.2.2 ARPPProcess Function	172
1.9.1.3.3 Internal Members	173
1.9.1.3.3.1 ARP_OPERATION_REQ Macro	173
1.9.1.3.3.2 ARP_OPERATION_RESP Macro	173
1.9.1.3.3.3 HW_ETHERNET Macro	173
1.9.1.3.3.4 ARP_IP Macro	173
1.9.1.4 BSD Sockets	174
1.9.1.4.1 Public Members	174
1.9.1.4.1.1 accept Function	175
1.9.1.4.1.2 AF_INET Macro	175
1.9.1.4.1.3 bind Function	175
1.9.1.4.1.4 BSDSocket Structure	175
1.9.1.4.1.5 closesocket Function	176
1.9.1.4.1.6 connect Function	176
1.9.1.4.1.7 gethostname Function	176
1.9.1.4.1.8 in_addr Structure	176
1.9.1.4.1.9 recv Function	177
1.9.1.4.1.10 INADDR_ANY Macro	177
1.9.1.4.1.11 recvfrom Function	178
1.9.1.4.1.12 INVALID_TCP_PORT Macro	178
1.9.1.4.1.13 send Function	178
1.9.1.4.1.14 IP_ADDR_ANY Macro	178
1.9.1.4.1.15 sendto Function	178
1.9.1.4.1.16 IPPROTO_IP Macro	179
1.9.1.4.1.17 IPPROTO_TCP Macro	179
1.9.1.4.1.18 IPPROTO_UDP Macro	179
1.9.1.4.1.19 listen Function	179

1.9.1.4.1.20 SOCK_DGRAM Macro	179
1.9.1.4.1.21 SOCK_STREAM Macro	180
1.9.1.4.1.22 sockaddr Structure	180
1.9.1.4.1.23 SOCKADDR Type	180
1.9.1.4.1.24 sockaddr_in Structure	180
1.9.1.4.1.25 SOCKADDR_IN Type	181
1.9.1.4.1.26 socket Function	181
1.9.1.4.1.27 SOCKET Type	181
1.9.1.4.1.28 SOCKET_CNXN_IN_PROGRESS Macro	181
1.9.1.4.1.29 SOCKET_DISCONNECTED Macro	181
1.9.1.4.1.30 SOCKET_ERROR Macro	182
1.9.1.4.2 Stack Members	182
1.9.1.4.2.1 BerkeleySocketInit Function	182
1.9.1.4.3 Internal Members	182
1.9.1.4.3.1 BSD_SCK_STATE Type	182
1.9.1.5 DNS	183
1.9.1.5.1 Public Members	183
1.9.1.5.1.1 DNSBeginUsage Function	183
1.9.1.5.1.2 DNSEndUsage Function	183
1.9.1.5.1.3 DNS_TYPE_A Macro	184
1.9.1.5.1.4 DNS_TYPE_MX Macro	184
1.9.1.5.2 Internal Members	184
1.9.1.5.2.1 DNSResolveROM Function	184
1.9.1.6 Dynamic DNS Client	184
1.9.1.6.1 Public Members	185
1.9.1.6.1.1 DDNS_POINTERS Structure	185
1.9.1.6.1.2 DDNS_SERVICES Enumeration	186
1.9.1.6.1.3 DDNS_STATUS Enumeration	187
1.9.1.6.1.4 DDNSClient Variable	188
1.9.1.6.1.5 DDNSForceUpdate Function	188
1.9.1.6.1.6 DDNSGetLastIP Function	188
1.9.1.6.1.7 DDNSGetLastStatus Function	188
1.9.1.6.1.8 DDNSSetService Function	189
1.9.1.6.2 Stack Members	189
1.9.1.6.2.1 DDNSInit Function	189
1.9.1.6.2.2 DDNSTask Function	189
1.9.1.6.3 Internal Members	189
1.9.1.6.3.1 DDNS_CHECKIP_SERVER Macro	190
1.9.1.6.3.2 DDNS_DEFAULT_PORT Macro	190
1.9.1.7 Hashes	190
1.9.1.7.1 Public Members	190
1.9.1.7.1.1 HashAddROMData Function	191

1.9.1.7.1.2 HASH_SUM Structure	191
1.9.1.7.2 Stack Members	191
1.9.1.7.2.1 MD5AddROMData Function	192
1.9.1.7.2.2 SHA1AddROMData Function	192
1.9.1.7.3 Internal Members	192
1.9.1.7.3.1 HASH_TYPE Enumeration	192
1.9.1.8 Helpers	193
1.9.1.8.1 Public Members	193
1.9.1.8.1.1 GenerateRandomDWORD Function	193
1.9.1.8.1.2 leftRotateDWORD Function	193
1.9.1.8.1.3 ROMStringToIPAddress Function	194
1.9.1.8.1.4 strnchr Function	194
1.9.1.8.1.5 ultoa Macro	194
1.9.1.9 HTTP2 Server	194
1.9.1.9.1 Features	195
1.9.1.9.1.1 Dynamic Variables	195
1.9.1.9.1.2 Form Processing	197
1.9.1.9.1.3 Authentication	200
1.9.1.9.1.4 Cookies	202
1.9.1.9.1.5 Compression	202
1.9.1.9.2 Public Members	203
1.9.1.9.2.1 curHTTP Variable	203
1.9.1.9.2.2 HTTP_CONN Structure	204
1.9.1.9.2.3 HTTP_IO_RESULT Enumeration	204
1.9.1.9.2.4 HTTP_READ_STATUS Enumeration	205
1.9.1.9.2.5 HTTPExecuteGet Function	205
1.9.1.9.2.6 HTTPExecutePost Function	206
1.9.1.9.2.7 HTTPReadPostPair Macro	207
1.9.1.9.2.8 sktHTTP Macro	208
1.9.1.9.3 Stack Members	208
1.9.1.9.3.1 HTTPInit Function	208
1.9.1.9.3.2 HTTPServer Function	208
1.9.1.9.4 Internal Members	208
1.9.1.9.4.1 curHTTPID Variable	209
1.9.1.9.4.2 HTTP_CACHE_LEN Macro	209
1.9.1.9.4.3 HTTP_FILE_TYPE Enumeration	210
1.9.1.9.4.4 HTTP_MAX_DATA_LEN Macro	210
1.9.1.9.4.5 HTTP_MIN_CALLBACK_FREE Macro	210
1.9.1.9.4.6 HTTP_PORT Macro	211
1.9.1.9.4.7 HTTP_STATUS Enumeration	211
1.9.1.9.4.8 HTTP_STUB Structure	212
1.9.1.9.4.9 HTTP_TIMEOUT Macro	212

1.9.1.9.4.10 HTTPGetROMArg Function	212
1.9.1.9.4.11 HTTPS_PORT Macro	212
1.9.1.9.4.12 httpStubs Variable	213
1.9.1.9.4.13 SM_HTTP2 Enumeration	213
1.9.1.9.4.14 RESERVED_HTTP_MEMORY Macro	213
1.9.1.10 ICMP	214
1.9.1.10.1 Public Members	214
1.9.1.10.1.1 ICMPBeginUsage Function	214
1.9.1.10.1.2 ICMPGetReply Function	214
1.9.1.10.1.3 ICMPEndUsage Function	215
1.9.1.10.1.4 ICMPSendPingToHostROM Macro	215
1.9.1.10.2 Internal Members	215
1.9.1.11 MPFS2	215
1.9.1.11.1 Public Members	216
1.9.1.11.1.1 MPFS_HANDLE Type	217
1.9.1.11.1.2 MPFS_INVALID Macro	217
1.9.1.11.1.3 MPFS_INVALID_HANDLE Macro	217
1.9.1.11.1.4 MPFS_SEEK_MODE Enumeration	217
1.9.1.11.1.5 MPFSClose Function	218
1.9.1.11.1.6 MPFSFormat Function	218
1.9.1.11.1.7 MPFSGetBytesRem Function	218
1.9.1.11.1.8 MPFSGetEndAddr Function	218
1.9.1.11.1.9 MPFSGetFlags Function	218
1.9.1.11.1.10 MPFSGetID Function	219
1.9.1.11.1.11 MPFSGetMicrotime Function	219
1.9.1.11.1.12 MPFSGetPosition Function	219
1.9.1.11.1.13 MPFSGetSize Function	219
1.9.1.11.1.14 MPFSGetStartAddr Function	219
1.9.1.11.1.15 MPFSGetTimestamp Function	220
1.9.1.11.2 Stack Members	220
1.9.1.11.2.1 MPFSInit Function	220
1.9.1.11.3 Internal Members	220
1.9.1.11.3.1 MPFS_PTR Type	221
1.9.1.11.3.2 MPFS_STUB Structure	221
1.9.1.11.3.3 MPFS_WRITE_PAGE_SIZE Macro	221
1.9.1.11.3.4 MPFS2_FLAG_HASINDEX Macro	221
1.9.1.11.3.5 MPFS2_FLAG_ISZIPPED Macro	222
1.9.1.11.3.6 MPFSOpenROM Function	222
1.9.1.11.3.7 MPFSTell Macro	222
1.9.1.11.3.8 MPFS_FAT_RECORD Structure	222
1.9.1.11.3.9 MPFS_INVALID_FAT Macro	223
1.9.1.12 NBNS	223

1.9.1.12.1 Stack Members	223
1.9.1.12.1.1 NBNSTask Function	223
1.9.1.13 Performance Tests	224
1.9.1.13.1 Stack Members	224
1.9.1.13.1.1 TCPPerformanceTask Function	224
1.9.1.13.1.2 UDPPerformanceTask Function	224
1.9.1.13.2 Internal Members	224
1.9.1.14 SMTP Client	225
1.9.1.14.1 Examples	225
1.9.1.14.1.1 Short Message	225
1.9.1.14.1.2 Long Message	226
1.9.1.14.2 Public Members	227
1.9.1.14.2.1 SMTP_CONNECT_ERROR Macro	227
1.9.1.14.2.2 SMTP_POINTERS Structure	228
1.9.1.14.2.3 SMTP_RESOLVE_ERROR Macro	229
1.9.1.14.2.4 SMTP_SUCCESS Macro	229
1.9.1.14.2.5 SMTPBeginUsage Function	230
1.9.1.14.2.6 SMTPClient Variable	230
1.9.1.14.2.7 SMTPEndUsage Function	230
1.9.1.14.2.8 SMTPFlush Function	230
1.9.1.14.2.9 SMTPIsBusy Function	230
1.9.1.14.2.10 SMTPIsPutReady Function	231
1.9.1.14.2.11 SMTPPutDone Function	231
1.9.1.14.2.12 SMTPSendMail Function	231
1.9.1.14.3 Stack Members	231
1.9.1.14.3.1 SMTPTask Function	231
1.9.1.14.4 Internal Members	232
1.9.1.14.4.1 SMTPPutROMArray Function	232
1.9.1.14.4.2 SMTPPutROMString Function	232
1.9.1.15 Reboot	232
1.9.1.15.1 Stack Members	233
1.9.1.15.1.1 RebootTask Function	233
1.9.1.16 RSA	233
1.9.1.16.1 Public Members	233
1.9.1.16.1.1 RSAEndUsage Function	234
1.9.1.16.1.2 RSAStep Function	234
1.9.1.16.1.3 RSA_DATA_FORMAT Enumeration	234
1.9.1.16.1.4 RSA_OP Enumeration	234
1.9.1.16.1.5 RSA_STATUS Enumeration	235
1.9.1.16.2 Stack Members	235
1.9.1.16.2.1 RSABeginDecrypt Macro	235
1.9.1.16.2.2 RSABeginEncrypt Macro	236

1.9.1.16.2.3 RSAEndDecrypt Macro	236
1.9.1.16.2.4 RSAEndEncrypt Macro	236
1.9.1.16.2.5 RSAInit Function	236
1.9.1.16.3 Internal Members	236
1.9.1.16.3.1 RSA_KEY_WORDS Macro	237
1.9.1.16.3.2 RSA_PRIME_WORDS Macro	237
1.9.1.16.3.3 SM_RSA Enumeration	237
1.9.1.17 SNTP Client	238
1.9.1.17.1 Public Members	238
1.9.1.17.1.1 SNTPGetUTCSeconds Function	238
1.9.1.17.2 Stack Members	238
1.9.1.17.2.1 SNTPClient Function	239
1.9.1.17.3 Internal Members	239
1.9.1.18 SSL	239
1.9.1.18.1 Generating Server Certificates	241
1.9.1.18.2 Public Members	242
1.9.1.18.2.1 SSL_INVALID_ID Macro	243
1.9.1.18.2.2 TCPSSLIsHandshaking Function	243
1.9.1.18.2.3 TCPIsSSL Function	243
1.9.1.18.2.4 SSL_SUPPLEMENTARY_DATA_TYPES Enumeration	243
1.9.1.18.2.5 SSL_PKEY_INFO Structure	243
1.9.1.18.2.6 SSL_RSA_CLIENT_SIZE Macro	244
1.9.1.18.3 Stack Members	244
1.9.1.18.3.1 SSL_STATE Type	244
1.9.1.18.3.2 SSLInit Function	245
1.9.1.18.3.3 TCPSSLGetPendingTxSize Function	245
1.9.1.18.3.4 TCPSSLHandleIncoming Function	245
1.9.1.18.3.5 TCPSSLHandshakeComplete Function	245
1.9.1.18.3.6 TCPStartSSLServer Function	245
1.9.1.18.3.7 SSL_MIN_SESSION_LIFETIME Macro	246
1.9.1.18.3.8 SSL_RSA_LIFETIME_EXTENSION Macro	246
1.9.1.18.4 Internal Members	246
1.9.1.18.4.1 RESERVED_SSL_MEMORY Macro	247
1.9.1.18.4.2 SM_SSL_RX_SERVER_HELLO Enumeration	247
1.9.1.18.4.3 SSL_ALERT Macro	248
1.9.1.18.4.4 SSL_ALERT_LEVEL Enumeration	248
1.9.1.18.4.5 SSL_APPLICATION Macro	248
1.9.1.18.4.6 SSL_BUFFER Union	248
1.9.1.18.4.7 SSL_BUFFER_SIZE Macro	249
1.9.1.18.4.8 SSL_BUFFER_SPACE Macro	249
1.9.1.18.4.9 SSL_CHANGE_CIPHER_SPEC Macro	249
1.9.1.18.4.10 SSL_HANDSHAKE Macro	249

1.9.1.18.4.11 SSL_HASH_SIZE Macro	250
1.9.1.18.4.12 SSL_HASH_SPACE Macro	250
1.9.1.18.4.13 SSL_KEYS Structure	250
1.9.1.18.4.14 SSL_KEYS_SIZE Macro	251
1.9.1.18.4.15 SSL_KEYS_SPACE Macro	251
1.9.1.18.4.16 SSL_MESSAGES Enumeration	251
1.9.1.18.4.17 SSL_SESSION Structure	252
1.9.1.18.4.18 SSL_SESSION_SIZE Macro	252
1.9.1.18.4.19 SSL_SESSION_SPACE Macro	253
1.9.1.18.4.20 SSL_SESSION_STUB Structure	253
1.9.1.18.4.21 SSL_SESSION_TYPE Enumeration	253
1.9.1.18.4.22 SSL_STUB Structure	254
1.9.1.18.4.23 SSL_STUB_SIZE Macro	255
1.9.1.18.4.24 SSL_STUB_SPACE Macro	255
1.9.1.18.4.25 SSL_VERSION Macro	255
1.9.1.18.4.26 SSL_VERSION_HI Macro	255
1.9.1.18.4.27 SSL_VERSION_LO Macro	256
1.9.1.18.4.28 SSLFinishPartialRecord Macro	256
1.9.1.18.4.29 SSLFlushPartialRecord Macro	256
1.9.1.18.4.30 SSLRxHandshake Function	256
1.9.1.19 TCP	256
1.9.1.19.1 Public Members	257
1.9.1.19.1.1 INVALID_SOCKET Macro	258
1.9.1.19.1.2 UNKNOWN_SOCKET Macro	258
1.9.1.19.1.3 TCP_ADJUST_GIVE_REST_TO_RX Macro	258
1.9.1.19.1.4 TCP_ADJUST_GIVE_REST_TO_TX Macro	259
1.9.1.19.1.5 TCP_ADJUST_PRESERVE_RX Macro	259
1.9.1.19.1.6 TCP_ADJUST_PRESERVE_TX Macro	259
1.9.1.19.1.7 TCP_OPEN_IP_ADDRESS Macro	259
1.9.1.19.1.8 TCP_OPEN_NODE_INFO Macro	259
1.9.1.19.1.9 TCP_OPEN_RAM_HOST Macro	260
1.9.1.19.1.10 TCP_OPEN_ROM_HOST Macro	260
1.9.1.19.1.11 TCP_OPEN_SERVER Macro	260
1.9.1.19.1.12 TCPConnect Macro	260
1.9.1.19.1.13 TCPClose Function	261
1.9.1.19.1.14 TCPDiscard Function	261
1.9.1.19.1.15 TCPDisconnect Function	261
1.9.1.19.1.16 TCPFind Macro	261
1.9.1.19.1.17 TCPFindArray Macro	261
1.9.1.19.1.18 TCPFindROMArray Macro	262
1.9.1.19.1.19 TCPFindROMArrayEx Macro	262
1.9.1.19.1.20 TCPFlush Function	262

1.9.1.19.1.21 TCPGetRemoteInfo Function	262
1.9.1.19.1.22 TCPGetRxFIFOFree Function	262
1.9.1.19.1.23 TCPGetRxFIFOFull Macro	263
1.9.1.19.1.24 TCPGetTxFIFOFree Macro	263
1.9.1.19.1.25 TCPGetTxFIFOFull Function	263
1.9.1.19.1.26 TCPIsConnected Function	263
1.9.1.19.1.27 TCPIsGetReady Function	263
1.9.1.19.1.28 TCPIsPutReady Function	264
1.9.1.19.1.29 TCPListen Macro	264
1.9.1.19.1.30 TCPPutROMArray Macro	264
1.9.1.19.1.31 TCPPutROMString Macro	264
1.9.1.19.1.32 TCPWasReset Function	264
1.9.1.19.2 Stack Members	265
1.9.1.19.2.1 SOCKET_INFO Type	265
1.9.1.19.2.2 TCB Type	265
1.9.1.19.2.3 TCB_STUB Type	265
1.9.1.19.2.4 TCP_SOCKET Type	266
1.9.1.19.2.5 TCP_STATE Type	266
1.9.1.19.2.6 TCPInit Function	266
1.9.1.19.2.7 TCPTick Function	266
1.9.1.19.3 Internal Members	266
1.9.1.20 Telnet	267
1.9.1.20.1 Public Members	267
1.9.1.20.2 Stack Members	267
1.9.1.20.2.1 TelnetTask Function	267
1.9.1.20.3 Internal Members	267
1.9.1.21 TFTP	268
1.9.1.21.1 Public Members	268
1.9.1.21.1.1 TFTPclose Macro	269
1.9.1.21.1.2 TFTPcloseFile Function	269
1.9.1.21.1.3 TFTPGet Function	270
1.9.1.21.1.4 TFTPGetError Macro	270
1.9.1.21.1.5 TFTPisFileClosed Function	270
1.9.1.21.1.6 TFTPisFileOpened Function	270
1.9.1.21.1.7 TFTPisFileOpenReady Macro	271
1.9.1.21.1.8 TFTPisGetReady Function	271
1.9.1.21.1.9 TFTPisOpened Function	271
1.9.1.21.1.10 TFTPisPutReady Function	271
1.9.1.21.1.11 TFTPOpen Function	272
1.9.1.21.1.12 TFTPOpenROMFile Macro	272
1.9.1.21.1.13 TFTP_ACCESS_ERROR Enumeration	272
1.9.1.21.1.14 TFTP_FILE_MODE Enumeration	272

1.9.1.21.1.15 TFTP_RESULT Enumeration	273
1.9.1.21.1.16 TFTPGetUploadStatus Function	273
1.9.1.21.1.17 TFTP_CHUNK_DESCRIPTOR Type	273
1.9.1.21.1.18 TFTP_UPLOAD_COMPLETE Macro	273
1.9.1.21.1.19 TFTP_UPLOAD_CONNECT Macro	273
1.9.1.21.1.20 TFTP_UPLOAD_CONNECT_TIMEOUT Macro	274
1.9.1.21.1.21 TFTP_UPLOAD_GET_DNS Macro	274
1.9.1.21.1.22 TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT Macro	274
1.9.1.21.1.23 TFTP_UPLOAD_RESOLVE_HOST Macro	274
1.9.1.21.1.24 TFTP_UPLOAD_SEND_DATA Macro	274
1.9.1.21.1.25 TFTP_UPLOAD_SEND_FILENAME Macro	275
1.9.1.21.1.26 TFTP_UPLOAD_SERVER_ERROR Macro	275
1.9.1.21.1.27 TFTP_UPLOAD_WAIT_FOR_CLOSURE Macro	275
1.9.1.21.2 Stack Members	275
1.9.1.21.2.1 TFTP_ARP_TIMEOUT_VAL Macro	275
1.9.1.21.2.2 TFTP_GET_TIMEOUT_VAL Macro	276
1.9.1.21.2.3 TFTP_MAX_RETRIES Macro	276
1.9.1.21.3 Internal Members	276
1.9.1.21.3.1 _tftpError Variable	276
1.9.1.21.3.2 _tftpSocket Variable	276
1.9.1.22 Tick	277
1.9.1.22.1 Public Members	277
1.9.1.22.1.1 TICK Variable	278
1.9.1.22.1.2 TICK_HOUR Macro	278
1.9.1.22.1.3 TICK_MINUTE Macro	278
1.9.1.22.1.4 TICK_SECOND Macro	278
1.9.1.22.1.5 TickGet Function	279
1.9.1.22.1.6 TickGetDiv256 Function	279
1.9.1.22.1.7 TickGetDiv64K Function	279
1.9.1.22.2 Stack Functions	279
1.9.1.22.2.1 TickInit Function	279
1.9.1.22.2.2 TickUpdate Function	280
1.9.1.22.3 Internal Members	280
1.9.1.22.3.1 TICKS_PER_SECOND Macro	280
1.9.1.23 UDP	280
1.9.1.23.1 Public Members	281
1.9.1.23.1.1 INVALID_UDP_PORT Macro	282
1.9.1.23.1.2 INVALID_UDP_SOCKET Macro	282
1.9.1.23.1.3 UDP_SOCKET Type	282
1.9.1.23.1.4 UDPOpen Macro	282
1.9.1.23.1.5 UDPClose Function	283
1.9.1.23.1.6 UDPDiscard Function	283

1.9.1.23.1.7 UDPFlush Function	283
1.9.1.23.1.8 UDPISGetReady Function	284
1.9.1.23.1.9 UDPISPutReady Function	284
1.9.1.23.1.10 UDPISOpened Function	284
1.9.1.23.1.11 UDP_OPEN_IP_ADDRESS Macro	284
1.9.1.23.1.12 UDP_OPEN_NODE_INFO Macro	284
1.9.1.23.1.13 UDP_OPEN_RAM_HOST Macro	285
1.9.1.23.1.14 UDP_OPEN_ROM_HOST Macro	285
1.9.1.23.1.15 UDP_OPEN_SERVER Macro	285
1.9.1.23.2 Stack Members	285
1.9.1.23.2.1 UDPInit Function	285
1.9.1.23.2.2 UDPTask Function	286
1.9.1.23.3 Internal Members	286
1.9.1.23.3.1 activeUDPSocket Variable	286
1.9.1.23.3.2 UDP_HEADER Type	286
1.9.1.23.3.3 UDP_PORT Type	287
1.9.1.23.3.4 UDP_SOCKET_INFO Type	287
1.9.1.23.3.5 UDPRxCount Variable	287
1.9.1.23.3.6 UDPSocketInfo Variable	287
1.9.1.23.3.7 UDPTxCount Variable	287
1.9.2 Wi-Fi API	288
1.9.2.1 Wi-Fi Network Topologies	290
1.9.2.1.1 Infrastructure Network	291
1.9.2.1.2 Ad-hoc Network	291
1.9.2.1.3 SoftAP Network	291
1.9.2.1.4 Wi-Fi Direct Network	294
1.9.2.2 Wi-Fi Connection Profile	294
1.9.2.2.1 Connection Profile Public Members	295
1.9.2.2.1.1 WFCPElementsStruct Structure	295
1.9.2.2.2 Connection Profile Internal Members	296
1.9.2.3 Wi-Fi Connection Algorithm	296
1.9.2.3.1 Connection Algorithm Public Members	296
1.9.2.3.1.1 WF_CAGetElements Function	297
1.9.2.3.1.2 WF_CASetElements Function	297
1.9.2.3.1.3 WFCAElementsStruct Structure	298
1.9.2.3.2 Connection Algorithm Internal Members	300
1.9.2.4 Wi-Fi Connection Manager	300
1.9.2.4.1 Connection Manager Public Members	300
1.9.2.4.1.1 WF_CMDDisconnect Function	301
1.9.2.4.1.2 WF_CMInfoGetFSMStats Function	301
1.9.2.4.1.3 WF_CMGetConnectContext Function	301
1.9.2.4.1.4 WF_DisableModuleConnectionManager Function	302

1.9.2.5 Wi-Fi Scan	302
1.9.2.5.1 Scan Operation and Scan Results	303
1.9.2.5.2 Shorter Scan or Connection Duration	304
1.9.2.5.3 Use of macro #define MY_DEFAULT_CHANNEL_LIST	305
1.9.2.5.4 Maximum Scan Results	305
1.9.2.5.5 Scan Public Members	305
1.9.2.5.5.1 tWFScanResult Structure	306
1.9.2.6 Wi-Fi Security	306
1.9.2.6.1 Wired Equivalent Privacy (WEP)	307
1.9.2.6.2 Wi-Fi Protected Access (WPA/WPA2)	308
1.9.2.6.3 Wi-Fi Protected Setup (WPS)	308
1.9.2.6.3.1 tWFWpsCred Structure	311
1.9.2.7 Wi-Fi Tx Power Control	312
1.9.2.7.1 Tx Power Control Public Members	312
1.9.2.8 Wi-Fi Power Save	312
1.9.2.8.1 Power Save Public Members	313
1.9.2.8.1.1 WF_HibernateEnable Function	313
1.9.2.8.1.2 WF_PsPollDisable Function	314
1.9.2.8.1.3 WFHibernate Structure	314
1.9.2.8.2 Power Save Internal Members	314
1.9.2.9 Wi-Fi Process Event	314
1.9.2.9.1 WiFi MRF24WG Events	317
1.9.2.10 Wi-Fi Miscellaneous	321
1.9.2.10.1 Wi-Fi Miscellaneous Public Members	322
1.9.2.10.1.1 WF_GetDeviceInfo Function	322
1.9.2.10.1.2 WF_GetMacStats Function	323
1.9.2.10.1.3 WF_EnableSWMultiCastFilter Function	323
1.9.2.10.1.4 WF_MulticastSetConfig Function	324
1.9.2.10.1.5 WFMacStatsStruct Structure	325
1.9.2.10.1.6 WFMulticastConfigStruct Structure	326
1.9.2.10.1.7 tWFDeviceInfoStruct Structure	326
1.9.2.11 Access Point Compatibility	326
1.9.2.12 802.11 AP/Router Configuration Settings	329
1.9.2.13 WiFi Troubleshooting Tips	330
1.9.2.13.1 Null String ESSID	330
1.9.2.13.2 Read back RF module Firmware version	330
1.9.2.13.3 RF Module Firmware Update	331
1.9.2.13.4 Wi-Fi Protected Setup (WPS) Issues	332
1.9.2.13.5 Network Switch or Change	333
1.9.2.13.6 Hibernate Mode	334
1.9.2.13.7 Management Scan Message Conflict	334
1.9.2.13.8 Handling of maximum length SSID	335

1.9.2.13.9 Multicast Filters : Hardware vs Software	337
1.9.2.13.10 MRF24WB0M assert failures whe using <iwconfig scan> command	337
1.9.2.13.11 MRF24WB0M advertised supported rates of 1, 2, 5.5 and 11 Mbps	339
1.9.2.13.12 MRF24WB0M Compatibility with AP/Routers	340
1.9.2.13.13 Encounter issues after upgrading MRF24WB0M RF module Firmware version 0x1207	340
1.9.2.13.14 How to fix MRF24WB0M / MRF24WG0M transmission rates	341
1.9.2.13.15 How to determine new IP address assigned	341
1.9.2.13.16 How to increase TCP throughput	342
1.9.2.13.17 Missing DHCP Client Name	343
1.9.2.13.18 Error Scenario And Possible Causes	345
1.9.2.14 Wireless Packets Analysis	347

Index

351

Microchip TCP/IP Stack Help

1 TCPIP Library

1.1 Introduction

Welcome to the Microchip TCP/IP Stack!

The Microchip TCP/IP Stack provides a foundation for embedded network applications by handling most of the interaction required between the physical network port and your application. It includes modules for several commonly used application layers, including HTTP for serving web pages, SMTP for sending e-mails, SNMP for providing status and control, Telnet, TFTP, Serial-to-Ethernet and much more. In addition, the stack includes light-weight and high-performance implementations of the TCP and UDP transport layers, as well as other supporting modules such as IP, ICMP, DHCP, ARP, and DNS.

This help file serves two purposes. The first is to be a guide for first-time users of the TCP/IP Stack. The Getting Started section begins a series of pages to help you become familiar with the stack and configure it for use on a Microchip development board.

The second purpose is to serve as a programmer's reference guide to the features and APIs available in the TCP/IP Stack.

Updates

The latest version of the Microchip TCP/IP Stack is always available at <http://www.microchip.com/tcpip>. New features are constantly being added, so check there periodically for updates and bug fixes.

Wi-Fi[®] is a registered trademark of the Wi-Fi Alliance.

1.1.1 Getting Help

Where to get help for the TCP/IP Stack

Description

The TCP/IP Stack is supported through Microchip's standard support channels. If you encounter difficulties, you may submit ticket requests at <http://support.microchip.com>

The Microchip forums are also an excellent source of information that can be accessed at <http://forum.microchip.com>.

Microchip also offers embedded network classes through Regional Training Centers. For more information, visit <http://www.microchip.com/rtc>.

1.2 Legal Information

This software distribution is controlled by the Legal Information at www.microchip.com/mla_license

1.3 Release Notes

Description

Microchip TCP/IP Stack Version Log:

Note: Please file bug-reports/bug-fixes at <http://support.microchip.com/> or post

to the Microchip TCP/IP -> Ethernet Forum at <http://forum.microchip.com/>

Look for stack updates at <http://www.microchip.com/mla/>

v5.45.00 August 2015

Changes:

1. Minor changes to fit latest version of XC8, XC16 and XC32 compilers.

v5.44.00 May 2015

Wi-Fi Changes:

1. For connection to AP in Infrastructure, WPS, or network redirection from SoftAP or Ad-Hoc mode, client station defaults to WPA-PSK/WPA2-PSK Auto when AP is in WPA-PSK, WPA2-PSK, WPA-PSK/WPA2-PSK Auto, or WPA-PSK/WPA2-PSK Mixed mode.
2. Improve DHCP Server to support up to 6 clients and recycle IP when lease expired or client disconnected.

Wi-Fi Notes:

1. MRF24WB WPA-PSK/WPA2-PSK Auto mode may not work when the target AP is set at WPA-PSK/WPA2-PSK Auto/Mixed mode. The solution is to set WPA-PSK or WPA2-PSK mode in AP.
2. Current Wi-Fi Demo Projects do not support MPLAB Code Configurator.

Wi-Fi G Demo Improvements:

1. DHCP Server

- support up to 6 clients
- recycle lease IP when client disconnected - enable/disable this feature with `#define CHECK_LINK_STATUS`
- longer ACK timeout to better accommodate busy wireless network

2. Wireless Security Mode

- client station uses WPA-PSK/WPA2-PSK Auto when connecting to AP in WPA-PSK, WPA2-PSK, or WPA-PSK/WAP2-PSK Auto/Mixed mode

Fixed mDNS indexes overrun bug to handle incoming message with length greater than 8 bits value.

v5.43.00 July 2014

Changes:

1. Ported WiFi support from legacy MLA.
2. SNMP support has not been ported.
3. SSL is no longer supported for PIC18 and XC8.
4. Improve OTA update. Uses .txt as well as .bin.
5. Improve WPA/WPA2 mixed mode behavior.
7. TCPIP Configuration Wizard utility has not been ported.

Fixes:

1. WiFi G demo scan lock-up problem.
2. Address TCP 0 length lock-up problem.
3. Improve RawMoveComplete lock-up problem.
4. Fix wrong RSSI indications.
5. Address SSL 2048 problem.

v5.42.08 June 2013

Changes:

1. Clarify item#4 in v5.42.04 Oct 2012 release. Change is applicable to MRF24WB/G only.
(MRF24WB/G) SSL client with RSA 2048bits is enabled by default. Applies only for PIC24/32. Not supported for PIC18.
2. (WiFi G Demo Board) Disable option WF_EASY_CONFIG_DEMO in WF_Config.h to resolve compilation error with lperfApplInit().

Fixes:

1. (MRF24WB) Resolve firmware 0x120C host scan bug : Host scan asserts when it is conducted after WPA-PSK fails due to key mismatch.

Refer to WFSan.c and WFMgmtMsg.c.

2. (MRF24WB) Resolve reset problem bug. Modified WF_SetCE_N and WF_SetRST_N to first set the level, than configure pin as output.

Refer to WFDriverPrv.h. This change is already available in WFDriverPrv_24G.h for MRF24WG in MLA v5.42.02 Aug 2012 release.

3. (MRF24W) Resolve WiFi Console demo IPERF issue bug fix : Enlarge generic tcp rx/tx buffer size. Refer to TCPIP MRF24W.h.

4. (MRF24W) Resolve EasyConfig bug fix for retry count when DISABLE_MODULE_FW_CONNECT_MANAGER_IN_INFRASTRUCTURE is enabled.

Refer to function WFEasyConfigProcess() in WFEasyconfig.c

5. (MRF24W) Resolve handling of max length SSID of AP. Refer to WFEasyConfig.c and CustomHTTPApp.c

v5.42.06 Feb 2013

Changes:

1. (MRF24WG) Due to memory constraints, all future RF module FW releases will follow this roll-out order

(Odd number eg 0x3109, 0x310b) Differences: Supports Wi-Fi Direct (P2P). But no EAP and no multi-client support.

(Even number eg 0x3108, 0x310a) Differences: Supports EAP (with MCHP approval) and multi-clients. But no Wi-Fi Direct support.

2. (MRF24WG) Added new project Wifi G demo. Wifi G demo currently only support softAP function,

3. (MRF24WG) In Wifi Console, for XC32-EX116-EAP_MRF24WG, supports security mode WF_SECURITY_WPA2_ENTERPRISE

(EAP-PEAP/MSCHAPv2 and EAP-TTLS/MSCHAPv2) .

Requires RF module FW version 0x3108 and even number releases. This requires special approval submitted to marketing.

4. (MRF24WG) Supports multi-client DHCP server (max 4 clients). SoftAP supports up to max 4 client

Requires RF module FW version 0x3108 and even number releases.

5. (MRF24WG) softAP supports active scan. Requires RF module FW version 0x3108 and future releases.

6. (MRF24WG) RF module FW version 0x3107 and future releases will NO longer support programming of regional domain via function

WF_SetRegionalDomain() due to changes in FCC requirements, which does not allow programming of the regional domain.

7. (MRF24WB/G) Only WEP key index 0 is valid for security mode WEP.

8.(MRF24WG) Created new event WF_EVENT_SOFT_AP_EVENT_SUBTYPE / WF_EVENT_SOFT_AP_EVENT, for softAP mode, to indicate client's

status (connected or disconnected to softAP). Requires RF module FW version 0x3108 and later releases. If prior

MLA releases are used with this new RF module FW version release, you need to port over this event handling in

WFEVentHandler.c and WF_ProcessEvent() (WiFi EZConfig).

9. Added full versions of RSA.c, ARCFOUR.c, and AES_PIC32MX.a to the TCP/IP Stack Distribution to support SSL and SNMPv3.

Previously these files were distributed in a separate cryptographic code distribution. These files are subject to the

U.S. Export Administration Regulations and other U.S. law, and may not be exported or re-exported to certain countries

or to persons or entities prohibited from receiving U.S. exports (including Denied Parties, entities on the Bureau of

Export Administration Entity List, and Specially Designated Nationals).

Fixes:

1. (MRF24WG) In EZConfig, SoftAP now supports Zeroconf & mDNS. Resolves wrong port issue.

Search for keyword SOFTAP_ZEROCONF_SUPPORT.

v5.42.04 Oct 2012

Changes:

1. (MRF24WB/G) Patch update features are added into MRF24WB0MA/B and MRF24WG0MA/MB.

MRF24WB0MA/B (no RF module FW version update is necessary)

Serial Port (Xmodem)

MRF24WG0MA/B (Requires RF module FW version 3107 or later.)

Serial Port (Xmodem), Web Client, Web Server

2. (MRF24WB/G) FTP client is added into WiFi console demo app.
3. (MRF24WB/G) Flexible scratch memory is used to allocate TCB.
4. SSL client with RSA 2048bits is enabled by default. Applies only for PIC24/32. Not supported for PIC18.
5. 16-bit mode feature is available on PIC32.

6. (MRF24WG) Enhance SoftAP with EZConfig features with pre-scan and redirection features.

Change SoftAP address from 192.168.1.1 to 192.168.1.3

Added WEP security, which requires RF module FW version 3107 or later.

Refer to WF_Config.h for more information.

7. Added support for PIC32MX6XX external PHY's: RTL 8201FL.

Only RMII configuration is supported with RTLPHY8201FL PHY.

This PHY driver works in Default Ethernet IO mode with PIC32MX675F512H .

If Alternate Ethernet IO mode is used for other PIC devices, then configuration bit need to be changed w.r.t Alternate Ethernet IO.

To get the status of MAC link with this PHY, PHY Link Status need to be read twice.

EthPhyGetLinkStatus(int refresh) API is used to read the PHY link status and for this PHY to read link status twice, the parameter refresh should be 1.

Fixes:

1. (MRF24WG) SoftAP under EZConfig has pre-scan and redirection features. Able to be redirected to infrastructure mode AP.

2. If SNMP_TRAP_DISABLED macro is enabled from TCPIP XXX.h file, there will be no TRAP table information for SNMP manager.

That is SNMP agent won't send information related to trap. By default SNMP_TRAP_DISABLED macro is disabled.

3. Fix typo error in v5.42.02

(MRF24WG) SoftAP default address is changed from 169.254.1.1 to 192.168.1.1

4. Fixed SNMP tabular issue if SNMP instances starts other than 0.
5. Fixed ARP initialization bug that resulted in TCP packets sent to ETH MAC broadcast address.

v5.42.02 Aug 2012

Fixes:

1. (MRF24WB/G) TCPIPConfig.exe is modified to support changes in macro definitions (CFG_WF_INFRASTRUCTURE, CFG_WF_ADHOC)

used in WF_Config.h.

2. (MRF24WB/G) Ad-hoc mode was left out in the TCPIP-Demo app. This mode is reinstated back into TCPIP-Demo app.

2 files are changed.

.\Microchip\Include\TCPIP Stack\WFDebugStrings.h

.\TCPIP\Demo App\WF_Config.h

3. (MRF24WB/G) SoftAP default address is changed from 192.168.1.1 to 169.254.1.1

4. (MRF24WG) Apply SSL fix.

Root cause is traced to be due to scratch pad memory to memory copy functions in WFMac_24G.c

5. (MRF24WG) Reset fix.

a. In file WFDriverCom_24G.c, MRF24WG reset sequence is modified.

- WFHardwareInit() does hard reset (putting I/O lines in correct state)

- ChipReset() does soft reset

b. In file WFDriverPrv_24G.h, macros WF_SetCE_N() and WF_SetRST_N() modified to first set the output level, then set the I/O direction

6. (MRF24WG) Applies to using PICDEMNet2 board and PIC18. Related to SPI interface settings.

a. In file WF_Spi.c, ConfigureSpiMRF24W() modified, for PIC18 only, to change the SPI clock idle state to low.

7. (MRF24WB/G) When host attempts to read scan results while the module FW is in reconnecting state, the module FW returns WF_ERROR_NO_STORED_BSS_DESCRIPTOR error. Reading scan results is only allowed in A) connected state; B) idle state.

Fix is in WaitForMgmtResponseAndReadData().

```
void WaitForMgmtResponseAndReadData(UINT8 expectedSubtype, UINT8 numDataBytes, UINT8 startIndex, UINT8
*p_data)
```

```
{ .....
```

```
/* check header result and subtype fields */
```

```
WF_ASSERT(hdr.result == WF_SUCCESS || hdr.result == WF_ERROR_NO_STORED_BSS_DESCRIPTOR);
```

```
WF_ASSERT(hdr.subtype == expectedSubtype);
```

```
.....
```

```
}
```

v5.42 Jul 2012

Changes:

1. All MPLAB 8 and MPLAB X projects have been modified to use the XC16 and XC32 compilers.
2. The colon character ":" has been added as a valid terminator for an IP address in the StringToIPAddress function.
3. (MRF24WB/G) PIC18 is only supported with "TCPIP Demo App".
MRF24WB0MA/MB works on PICDEM PIC18 Explorer.
MRF24WG0MA/MB works on PICDemNet2 PIC 18 but has issues with PICDEM PIC18 Explorer.
4. (MRF24WB/G) Enhanced debug messages providing more details such as authentication or association failures, etc.
5. (MRF24WB/G) Added option to disable module FW connection manager by adding "#define DISABLE_MODULE _FW_CONNECT_MANAGER_IN_INFRASTRUCTURE".
6. (MRF24WB/G) Added option to derive real key from pass-phrase in host side by adding "#define DERIVE _KEY_FROM_PASSPHRASE_IN_HOST".
7. (MRF24WB/G) Enabled WEP security for EasyConfig Ad-hoc connection
8. (MRF24WB/G) Added support for MRF24WG0MA/MB, a superset of MRF24WB0MA/MB (MRF24WB).
Required to add to project definition files (#define MRF24WG).
Ranges are different from MRF24WB0MA/MB.
 - i. Valid RSSI Range : 43 ~ 128 (max) (WF_ScanGetResult())
 - ii. Max transmit power range : 0 ~ 18 dbm (WF_TxPowerSetMax())

MRF24WG0MA/MB enhancements are above and beyond on those features listed above.

- a. Added WPS(1.0) security method (#define WF_SECURITY_WPS_PUSH_BUTTON & WF_SECURITY_WPS_PIN)
(TCPIP demo + console demo).
- b. Added Wi-Fi Direct (P2P) function in the GC (group client) role
(#define MY_DEFAULT_NETWORK_TYPE CFG_WF_P2P)
(TCPIP demo + console demo)
- c. Added simplified basic SoftAP functionality (WF_SOFT_AP).
Current features are 1 client STA, open security and no routing.
(Easy config demo)
- d. Selection of support for 16 (max) software multicast filters (ENABLE_SOFTWARE_MULTICAST_FILTER)
or 2 hardware multicast filters. RTS or ES release requires software patch.

Fixes:

1. The BigInt_helper.S file will now correctly include the processor include files for the PIC24E and dsPIC33E architectures.
2. Fixed bug in TCP state machine that corrupted the sequence number when client socket is closed.
3. Fixed bug in BerkeleyAPI.c::sendto() that checked for the wrong UDP socket to be opened.
4. (All parts including RTS or ES release) MRF24WG0MA/MB requires PLL work-around initialization code.
Please refer to July/2012 release.

5. (All parts including RTS or ES release) MRF24WG0MA/MB needs to update stack from July/2012 release to cater for SSL.

6. (MRF24WB/G) Fixed stack issues encountered in Stack version v2012-04-03 and older.

Code fix is needed in WF_CPSetElements() API. Modification is as below.

```
void WF_CPSetElements(UINT8 CpId, tWFCPElements *p_elements) {
    UINT8 elements[sizeof(tWFCPElements) + 2];
    WF_ASSERT(p_elements->ssidLength <= WF_MAX_SSID_LENGTH);
    memset(elements, 0, sizeof(elements));
    memcpy(elements, p_elements, sizeof(*p_elements));
    LowLevel_CPSetElement(CpId, /* CP ID */
        WF_CP_ELEMENT_ALL, /* Element ID */
        (UINT8 *)elements, /* pointer to element data */
        sizeof(elements)); /* number of element data bytes */
}
```

Code fix is also needed in WaitForMgmtResponse() API (WFMgmtMsg.c). Refer to description of this fix is in the Supplementary TCPIP Help folder.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenabling its DHCP server.
3. LCDBlocking.c timing for the LCD_E_IO enable signal is too fast to meet published data sheet limits for various LCD controllers when compiling for a PIC32 running at > 43MHz. Despite this potential timing violation, the LCD does normally work correctly on the precompiled PIC32 demos for the Explorer 16 at 80MHz.
4. For PIC32 implementations, depending on the configuration, it is possible that the MACGetFreeRxSize() returns a value greater than 64 KB.
For backward compatibility reasons the stack uses a 16 bit value to check the returned value

and it won't work correctly.

5. Limiting the number of UDP sockets to 8 in the stack demos may prevent SNMP trap functionality. If this occurs, you can increase the MAX_UDP_SOCKETS definition in TCPIPConfig.h to 10 (if your system will support the increased data memory usage) to fix this issue.

6. The SNMP mib file's date and version parameter does not match the date/version of the current stack release.

7. (MRF24WG0MA/MB) RTS or ES release requires software patch for multicast filters.

8. (MRF24WG0MA/MB) Under softap option with easy config demo, unable to connect to infrastructure mode AP.

9. Certain APs have power save issue with reconnect. Please refer to AP compatibility list.

v5.41.02 Apr 2012

Fixes:

1. Added the Bigint helper libraries for C30 into the stack source folder to prevent linker errors in 16-bit projects that include those libraries.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.

2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenabling its DHCP server.

3. LCDBlocking.c timing for the LCD_E_IO enable signal is too fast to meet published data sheet limits for various LCD controllers when compiling for a PIC32 running at > 43MHz. Despite this potential timing violation, the LCD does normally work correctly on the precompiled PIC32 demos for the Explorer 16 at 80MHz.

4. For PIC32 implementations, depending on the configuration, it is possible that the MACGetFreeRxSize() returns a value greater than 64 KB.

For backward compatibility reasons the stack uses a 16 bit value to check the returned value and it won't work correctly.

5. Limiting the number of UDP sockets to 8 in the stack demos may prevent SNMP trap functionality. If this occurs, you can increase the MAX_UDP_SOCKETS definition in TCPIPConfig.h to 10 (if your system will support the increased data memory usage) to fix this issue.

6. The SNMP mib file's date and version parameter does not match the date/version of the current stack release.

v5.41 Feb 2012

Changes:

1. Added socket number validation to TCP functions.
2. Removed the WiFi projects from the Energy Monitoring and WebVend demos.
3. Added the "-mperipheral-libs" linker option to C32 projects.
4. Moved AES library and header files used by SNMPv3 from the "Crypto" source and include folders to the to the "TCPIP Stack" source and include folders.
5. Changed SNMP MIB trap number from 7 to 0.
6. Changed SNMP MIB SYOBJECTID to 43.6.1.4.1.17095.1 (added .1).
7. Updated MPFS2 Utility to make path settings persistent.
8. Changed default channel list from {1,6,11} to {1,2,3,4,5,6,7,8,9,10,11} for FCC domain
9. Added support for WEP with Shared Key
10. Added new command "iwconfig scan" & "iwconfig scanresults"
11. Added gRFModuleVer1209orLater" flag to identify version 1209 and later specific API's.
12. Added the following 1209 and later specific API:
 - a) WF_CPSetWepKeyType()
 - b) WF_CMGetConnectContext()
 - c) WFEnableBroadcastProbeResponse()
 - d) WFEnableAggressivePowerSave()
 - e) WF_CPSetSsidType()
 - f) WFEnableDeferredPowerSave()
 - g) WF_FixTxRateWithMaxPower()

Fixes:

1. Updated MPFS2 Utility path strings for Mac/Linux compatibility.
2. Iperf - Resolved application crash due to network disconnection.
3. Host Scan - More robust and prevent system hang up. #define WF_HOST_SCAN.

4. Power Save - Improvements with Aggressive PS mode, and better handling of PS in host code.
5. Host connect & disconnect - Prevent system hang.
6. DHCP Refresh - Better handling of DHCP session in PS mode, and issue DHCP renewal any STA reconnect with AP.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.
3. LCDBlocking.c timing for the LCD_E_IO enable signal is too fast to meet published data sheet limits for various LCD controllers when compiling for a PIC32 running at > 43MHz. Despite this potential timing violation, the LCD does normally work correctly on the precompiled PIC32 demos for the Explorer 16 at 80MHz.
4. For PIC32 implementations, depending on the configuration, it is possible that the MACGetFreeRxSize() returns a value greater than 64 KB.
For backward compatibility reasons the stack uses a 16 bit value to check the returned value and it won't work correctly.
5. Limiting the number of UDP sockets to 8 in the stack demos may prevent SNMP trap functionality. If this occurs, you can increase the MAX_UDP_SOCKETS definition in TCPIPConfig.h to 10 (if your system will support the increased data memory usage) to fix this issue.
6. The SNMP mib file's date and version parameter does not match the date/version of the current stack release.

v5.36.4 Oct 2011

Changes:

1. SNMPNotify() is updated to support ASCII string variable type for

both TRAPv1 and TRAPv2.ASCII string address pointer is assigned to argument val(SNMP_VAL) of SNMPNotify().

2. The SSL module has been updated to support 1024-bit RSA key lengths for server and client on all architectures. PIC32 microcontrollers now support client/server RSA key lengths of 2048 bits. NOTE: To support these changes, you must manually modify your copy of RSA.c. A description of the required changes ("Required SSL Changes.pdf") can be found in your Microchip Applications Libraries installation directory in the "\Microchip\Help\Supplementary TCPIP Help" subdirectory.

Fixes:

1. SNMP local variable community length increased with plus one.SNMP warnings has been removed for the compiler version C32 2.01 for zero optimisation.
2. Updated MPFS2.jar and mib2bib.jar to support Java version 1.7.
3. Fixed MPFS2.jar offset issue for fileRcrd.bin and dynRcrd.bin file and it was due to the file which has zero dynamic variable.Fixed Crimson editor problem with webPage2 folder where user couldn't save files using Crimson Editor if the WebPages2 folder that contained those files was selected in the MPFS2 utility.
4. MPFS2.jar file was getting hanged for the zero file size access.Now Zero file size also is the part of the respective generated files.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.
3. LCDBlocking.c timing for the LCD_E_IO enable signal is too fast to meet published data sheet limits for various LCD controllers when compiling for a PIC32 running at > 43MHz. Despite this potential timing violation, the LCD does normally work correctly on the precompiled PIC32 demos for the Explorer 16 at 80MHz.
4. For PIC32 implementations, depending on the configuration, it is possible that the

MACGetFreeRxSize() returns a value greater than 64 KB.

For backward compatibility reasons the stack uses a 16 bit value to check the returned value and it won't work correctly.

5. Limiting the number of UDP sockets to 8 in the stack demos may prevent SNMP trap functionality. If this occurs, you can increase the MAX_UDP_SOCKETS definition in TCPIPConfig.h to 10 (if your system will support the increased data memory usage) to fix this issue.

6. The SNMP mib file's date and version parameter does not match the date/version of the current stack release.

v5.36.2 July 2011

Changes:

1. Removed the Google PowerMeter and Google PowerMeter EZConfig demos. Google, Inc. has deprecated Google PowerMeter and has expressed the intent to remove access to it on September 16, 2011. To obtain Microchip Technology's Google PowerMeter reference implementation, you can download the June 2011 Microchip Application Libraries archived release from www.microchip.com/mal.
2. Modified the Energy Monitoring demo to remove Google PowerMeter functionality. The demo will still display measured power data on its internal web page.
3. Updated the TCP/IP Stack Performance table to use the testing methodology from previous releases. More information is available in the TCP/IP Stack Help file.
4. gSnmplibNonMibRecInfo[] has been moved from snmp.c file to CustomSNMPApp.c file and SNMP_MAX_NON_REC_ID_OID macro has been moved from snmp.h file to CustomSNMPApp.c file. gSnmplibNonMibRecInfo[] is used to list the static variables parent OID strings which are not part of mib.h file. This structure is used to restrict the access to the SNMPv3 objects from SNMPv2c and SNMPv1 version requests. Macro STACK_USE_SMIV2 is used to support gSnmplibNonMibRecInfo[] with MODULE-IDENTITY number. For V5.31 STACK_USE_SMIV2 need to commented.
5. Removed the SPI2CON register freeze-on-halt bit macro from the SPIFlash, RAM, and EEPROM driver files to provide compatibility with C32 v2.00.

Fixes:

1. Removed the MPFSImg2 files from the MPLAB X C18/C30 projects so that the projects will compile. Disabled MPFSImg2.c for PIC32 Explorer 16 projects.
2. Added a heap and minimum stack size for the PIC32 Ethernet Starter Kit MPLAB X project.
3. The TCP/IP Stack Help File's performance table has been updated using the same

test procedure used in previous releases.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.
3. LCDBlocking.c timing for the LCD_E_IO enable signal is too fast to meet published data sheet limits for various LCD controllers when compiling for a PIC32 running at > 43MHz. Despite this potential timing violation, the LCD does normally work correctly on the precompiled PIC32 demos for the Explorer 16 at 80MHz.
4. For PIC32 implementations, depending on the configuration, it is possible that the MACGetFreeRxSize() returns a value greater than 64 KB. For backward compatibility reasons the stack uses a 16 bit value to check the returned value and it won't work correctly.
5. Limiting the number of UDP sockets to 8 in the stack demos may prevent SNMP trap functionality. If this occurs, you can increase the MAX_UDP_SOCKETS definition in TCPIPConfig.h to 10 (if your system will support the increased data memory usage) to fix this issue.
6. The SNMP mib file's date and version parameter does not match the date/version of the current stack release.

v5.36 2 June 2011

Changes:

1. Because of changes to the SHOUTcast server configuration, the Internet Radio demo is no longer functional. This demo has been retained in the stack distribution to provide a TCP/IP code example.
2. The UDP module will now perform address resolution and DNS querying automatically. As a result, the UDP module APIs have changed. The UDPOpenEx function provides this

additional functionality. Please consult the TCP/IP Stack Help File's

"Stack API > UDP" topic for more information.

3. The UDPOpen macro has been added to conform to the legacy UDPOpen interface.

4. The Announce, BerkeleyAPI, DHCP client/server, DNS client/server, NBNS, Reboot, SNMP, SNTP, TFTPc, UDPPerformanceTest, and ZeroConf modules have been updated to use the new UDP API. The iPerf demo has also been updated.

5. The MPFS Classic and HTTP(1) modules have been removed from the stack. Functionality to support these modules has also been removed from the TCP/IP Stack software tools. MPFS2 and HTTP2 are still supported.

6. The UARTConfig demo module has been updated to upload MPFS2 images to the demo board in place of MPFS Classic images.

7. To facilitate linking on PIC18 platforms, the number of UDP sockets in demo projects has been reduced from 10 to 8.

8. The SNMP Stack application and mib2bib.jar PC utility now both support 1024 dynamic IDs.

9. SNMP_DEMO_TRAP is a new dynamic variable added to the snmp.mib file to support SMIv2 with TRAPv2. This will correct a previously existing issue viewing traps with the iReasoning MIB browser. As per those changes, the mchp.mib file has been modified to support the SMIv2 standard. This mib includes MODULE-IDENTITY which will provide MICROCHIP and MIB information. snmp.mib also includes MODULE-IDENTITY(1), a new number (1) to the existing OID after ENTERPRISE-ID(17095).

10. Added a preprocessor check that will include the ultoa function if a version of the C32 compiler earlier than 1.12 is used.

11. Modified the WiFi module to use separate retry counters for AdHoc and Infrastructure modes.

12. Modified Berkeley API module to accept IPPROTO_IP as a valid protocol for the socket function. The code will determine the protocol type from the socket type (datagram or stream).

13. Created MPLAB X projects corresponding to most MPLAB 8 projects and configurations. These projects are located in the MPLAB.X subfolder in the associated demo project directory. The MPLAB X import wizard can be used to create MPLAB X projects from MPLAB 8 projects that don't have an analogue in the new demo project folders.

14. Added project support for the dsPIC33E and PIC24E architectures.

15. All TCP/IP Stack demo projects have been moved to the "TCPIP" subdirectory in the stack installation directory.

16. Created Java versions of several TCP/IP tools to provide cross-platform support. The TCP/IP Configuration Wizard has not been ported to Java; thus, it is only available for Windows users.

17. To prevent issues with path length, MPLAB 8 project names have been changed.

A list of the abbreviations used in the project names is available in the MAL help folder (Microchip Solutions/Microchip/Help/Abbreviations.htm). The names of the HardwareProfile and TCPIPConfig configuration files have been abbreviated as well.

18. Changed the configuration inclusion macros used by the TCP/IP Stack demo projects to match the terms used in the project/configuration names.

19. The "Alternative Configurations" subfolders in most demo projects has been renamed to "Configs."

20. Added a Google PowerMeter demo for use with the PIC18F87J72 Energy Monitoring PICtail.

21. The Web Preview tool is no longer included with the stack.

Fixes:

1. Fixed a DHCP Server (DHCPs.c) lease leak problem that would occur when STACK_USE_ZEROCONF_LINK_LOCAL was defined. This problem would have resulted in the DHCP server stop giving out any leases until being rebooted.
2. Updated the PIC32MX6XX/7XX external PHY SMSC 8720LAN reference design.
3. Fixed bug with window expecting MACGetFreeRxSize() to return values < 32KB.
4. Fixed a type casting bug with the CalcIPChecksum function that would cause an incorrect TX checksum if the checksum value overflowed again after adding the carry bits to the checksum value.
5. Fixed a bug in the AutoIP module that may have prevented the module from correctly defending its own address.
6. Added a check to the Announce module to ensure the MAC layer is linked before attempting to transmit an Announce message.
7. Fixed a bug in the ETH97J60 MACPut function.
8. Added an additional preprocessor check in a debug menu setting in WF_Spi.c to prevent a build error.
9. Added a fix to the Google PowerMeter demo code to restore SNTP timestamp sourcing if SNTP is enabled. Previously, it would be overwritten by a possibly invalid HTTP timestamp.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via

a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.

3. LCDBlocking.c timing for the LCD_E_IO enable signal is too fast to meet published data sheet limits for various LCD controllers when compiling for a PIC32 running at > 43MHz. Despite this potential timing violation, the LCD does normally work correctly on the precompiled PIC32 demos for the Explorer 16 at 80MHz.

4. For PIC32 implementations, depending on the configuration, it is possible that the MACGetFreeRxSize() returns a value greater than 64 KB.

For backward compatibility reasons the stack uses a 16 bit value to check the returned value and it won't work correctly.

5. Limiting the number of UDP sockets to 8 in the stack demos may prevent SNMP trap functionality. If this occurs, you can increase the MAX_UDP_SOCKETS definition in TCPIPConfig.h to 10 (if your system will support the increased data memory usage) to fix this issue.

6. The SNMP mib file's date and version parameter does not match the date/version of the current stack release.

v5.31 19 October 2010

Changes:

1. Reorganized demo projects. The TCPIP ENC24J600 Demo App, TCPIP PIC32 ETH Demo App, and TCPIP WiFi Demo App projects in stack versions 5.25 and prior have all been merged into the TCPIP Demo App folder. All four of these projects were almost identical to each other, with the primary difference being the network interface controller the projects were preconfigured to support. In this 5.31 TCP/IP Stack release, each hardware combination now has its own MPLAB IDE project in the TCPIP Demo App folder.
2. Reorganized HardwareProfile.h and TCPIPConfig.h structure for the TCPIP Demo App, TCPIP Google PowerMeter Demo App, and TCPIP WebVend App projects. Now, instead of having one massive HardwareProfile.h file that supports a multitude of hardware platforms simultaneously, simple individual hardware profiles have been created for specific hardware combinations and placed in the "Alternative Configurations" sub-folder. The correct hardware profile or TCPIP config file is selected by #including "HardwareProfile.h"

or "TCPIPConfig.h" as in previous stack versions. However, the active hardware profile or config file from the Alternative Configurations folder is selected by a preprocessor macro defined in the MPLAB project settings (passed on the command line to the compiler).

3. Added HTTP_SAVE_CONTEXT_IN_PIC_RAM option to TCPIPConfig.h (HTTP2 server module). This option, when enabled, will increase HTTP2 server performance when servicing multiple simultaneous connections at the expense of using more PIC RAM.

4. Added automatic TPIN+/- polarity swapping logic to ETH97J60.c driver for PIC18F97J60 family devices. Some 3rd party Ethernet switches, routers, and end devices have their TX+/- pins wired backwards such that the remote TX+ signal connects to the PIC TPIN- pin and the remote TX- signal connects to the PIC TPIN+ pin. Because 10BaseT Ethernet signaling is polarized and the PIC18F97J60 family does not implement an auto-polarity feature, this normally prevents all RX communications with these non-IEEE 802.3 compliant link partners. To work around this incompatibility, it is possible to implement circuitry to swap the RX+ and RX- signals before reaching the TPIN+ and TPIN- pins. The PICDEM.net 2 Rev 6 reference design includes this necessary circuitry (U6, U7, R54, and RX_SWAP GPIO output pin from PIC). This stack version automatically controls the RX_SWAP signal based on the ETH_RX_POLARITY_SWAP_TRIS and ETH_RX_POLARITY_SWAP_IO definitions in HardwareProfile.h. If these macros are undefined, then the automatic polarity swapping feature is disabled in the ETH97J60.c driver.

5. Added portable LFSRRand() and LFSRSeedRand() public functions to Helpers.c and removed all references to C rand() and srand() functions. The C rand() function returns a 15 bit integer on 8 and 16 bit PICs, but a 31 bit integer on PIC32s. The LFSRRand() function returns a 16-bit integer, regardless of which PIC you are using.

6. Added support for various SST/Microchip brand SST25xxxx SPI Flash chips to SPIFlash.c driver. Previously only devices supporting the Auto Address Increment (AAI) Word Program opcode (0xAD) would work. Now, devices such as the SST2525VF010A should work, which require the AAI Byte program opcode (0xAF) instead.

7. Removed support for Spansion brand SPI Flash chips in the SPIFlash.c driver.

If your application is already using one of these devices, continue to use the SPIFlash.c/.h files from TCP/IP Stack 5.25. These older files are API compatible with the current version, so can be dropped in by simply overwriting the SPIFlash.c and SPIFlash.h files.

8. Removed a -4 offset from the advertised TCP Maximum Segment Size option (MSS) and TCP_MAX_SEG_SIZE_RX configuration value in TCP.c. The default TCP MSS advertised is now 536 instead of 532.
9. For Wi-Fi projects in the TCPIP Demo App folder, changed MY_DEFAULT_LIST_RETRY_COUNT to WF_RETRY_FOREVER instead of 3. This changes default connection behavior to keep trying to connect instead of just trying 3 times which makes more sense for demonstration.
10. Changed WF_Connect() beacon timeout to 40.
11. IFConfig command in TCPIP WiFi Console Demo App modified to return application-perspective MAC address from the AppConfig structure, and not the Wi-Fi serialized MAC address (they may not match if user desired custom MAC).
12. Updated the TCP/IP Configuration Wizard. The user can now configure wireless settings and stack settings separately. Because of the changes to the TCPIPConfig.h file, the user must now select the specific copy of TCPIPConfig.h (or any of its variants) instead of selecting a project directory. Added the ability to select WF_RETRY_FOREVER in the Wi-Fi configuration settings. Added a selection parameter for BSD socket count. Added validation to check for the proper number of Berkeley sockets and TCP performance test sockets in the socket configuration screen (Advanced Settings) if either of these features are enabled. Added the ability to create sockets of the same type with different TX/RX buffer sizes in the socket configuration screen.
13. Updated the TCPIP WebVend Demo App to support Wi-Fi in several configurations.
14. Modified the Google PowerMeter demo to automatically determine the date/time from the HTTP module if the date/time cannot be obtained from the SNTP module.
15. Added a new Google Map project example to the Combo Demos folder. This example runs on a PIC24FJ256DA210 Development Board + Fast 100Mbps Ethernet PICtail Plus (or Ethernet PICtail Plus) + Truly 3.2" 240x320 display, TFT_G240320LTSW_118W_E (or Powertip 4.3" 480x272 display, PH480272T_005_I11Q). It also can run on the PIC32 Multimedia Expansion Board + PIC32 Ethernet Starter Kit. This demo connects to the Internet, sends an HTTP query for a specific map tile to the Google Static Maps API, and then displays the compressed tile to the graphics display. For more information, see the "Getting Started - Running the Graphics Google Map Demo.htm" file in the Combo Demos\Google Map folder.

16. Added preliminary SNMPv3 module. This module, enabled with the `STACK_USE_SNMPV3_SERVER` option in `TCPIPConfig.h`, implements the Simple Network Management Protocol, version 3. Among other things, SNMPv3 adds secure authentication and cryptographic privacy as compared to SNMPv2C. This implementation currently only supports AES encryption (no DES support). It also has only been tested with the PIC32 Ethernet Starter Kit (TCPIP Demo App - C32 - `PIC32_ENET_SK_DM320004_INTERNAL_ETHERNET.mcp` MPLAB IDE project). SNMPv3 on PIC18, PIC24, and dsPIC platforms are not supported at this time. Because AES encryption has specialized United States export requirements, this TCP/IP Stack release does not include the required AES library to enable SNMPv3. To obtain the needed AES library, you must purchase SW300052 v2.6 or later. Older v2.5 and previous versions include AES related files on them, but do not include the new AES files required by SNMPv3. For more information on using SNMP, refer to the TCP/IP Stack Help (Demo Information -> Available Demos -> TCPIP Demo App -> Demo Modules -> Network Management (SNMP) Server).

17. Altered the `SaveAppConfig()` function in `MainDemo.c` to store a more robust signature to EEPROM/SPI Flash when saving the `AppConfig` structure. In v5.25 and prior stack versions, when EEPROM or SPI Flash memory was available, the stack would automatically write a one byte marker character to address `0x000000` in the EEPROM/Flash indicating if a valid `AppConfig` structure was stored in the non-volatile memory. This resulted in the EEPROM/Flash contents being organized like the following:

Address Data Contents

=====

0x000000: Marker Byte

0x000001: AppConfig structure

MPFS_RESERVE_BLOCK: Start of MPFS/MPFS2 image containing web pages

In this stack version, EEPROM/Flash contents will now contain:

Address Data Contents

=====

0x000000: Length of AppConfig structure (16-bit integer)

0x000002: IP checksum of the AppConfig default values, as defined in `TCPIPConfig.h` and `WF_Config.h` (16-bit integer).

0x000004: IP checksum of the subsequent EEPROM/Flash bytes of the AppConfig values.

0x000006: AppConfig structure

MPFS_RESERVE_BLOCK: Start of MPFS/MPFS2 image containing web pages

The additional checksums allow automatic detection to occur if you change one of the values in TCIPConfig.h or WF_Config.h that affects AppConfig. If you change one of the values in code, then upon boot up, the application will automatically detect this change and start using the values that you selected in code. If, at run time, you decide to change the AppConfig values and commit the changes to EEPROM/Flash, then the stack will subsequently use the run-time saved values on future reboots. The checksum at offset 0x000004 ensures that if any corrupted AppConfig contents are found in EEPROM/Flash (ex: power is lost between writing the signature structure and actual AppConfig structure, or code unintentionally overwrites something in the AppConfig memory area), then the original defaults defined in TCIPConfig.h and WF_Config.h will be used instead of the corrupted values. This EEPROM/SPI Flash change affects all projects except TCIP Internet Radio App, TCIP Internet Bootloader App, and all PIC32 Starter Kit projects since these projects do not have or use external EEPROM or SPI Flash memory.

Fixes:

1. Fixed a UDP bug in which a transmitted packet would have been addressed to the wrong destination node if the UDP socket received a broadcast packet from a different remote node from the last received packet, but using the same source port number as the last received packet. The FindMatchingSocket() function in UDP.c will now always change the local socket parameters to send to the most recent remote node's unicast IP address, regardless of if the last received packet was addressed to a multicast or broadcast destination. Thanks go to Billy Walton for reporting this erroneous behavior. If you wish to change the destination IP/MAC addresses or port number for a UDP packet that you are ready to send, write the new parameters to the UDPSocketInfo[SocketHandle] global structure before calling UDPFlush(). This structure contains remoteNode and remotePort parameters for the remote IP address/MAC address and remote UDP port, respectively. You can also read these values to obtain the remote addressing parameters for the last received packet on the given UDP socket. Note that "SocketHandle" refers to the UDP socket handle returned by the UDPOpen() API, not the literal string "SocketHandle".
2. Fixed ADC state save/restore bug in GenerateRandomDWORD() function in Helpers.c. PIC24, dsPIC, and PIC32 platforms require the ADC ON/ADON bit to

be cleared before modifying certain other ADC register contents.

3. Fixed an ENC28J60 MAC/MII register write timing violation when using a PIC24H or dsPIC at over 33MIPS. There was inadequate Chip Select hold time provided, violating the 210ns minimum specified in the ENC28J60 data sheet. This violation may have resulted in certain devices losing the ability to receive packets (due to the MARXEN bit, MACON1<0>, getting cleared unintentionally).

4. Fixed an ENC24J600.c driver bug in which operating at 100Mbps with the ENC424J600/624J600 Ethernet controller, it would be possible for the MACGetHeader() function to issue a Reset() operation under rare circumstances. The PIC would reset whenever the PHY detected an illegal symbol during 4B5B decoding but guessed the correct 4B symbol such that no data corruption or CRC error occurred. This condition results in a valid packet being received but with the Received Ok Receive Status Vector bit being clear (RSV<23> == 0). This issue would become more probable when using very long Ethernet cables (ex: 100 meters) and receiving a lot of data.

5. Fixed a TCP bug in which calling TCPDisconnect() to close a connection when the remote node's RX window was 0 bytes would have caused the stack to enter an infinite loop sending duplicate ACK packets.

6. Fixed Wi-Fi bug that caused assert condition if too many management messages were being received during data traffic.

7. Fixed Wi-Fi bug that caused WF_EVENT_CONNECTION_REESTABLISHED event case to send the wrong notification to the app.

8. Fixed Wi-Fi bug that caused assert failure with Scratch move failure.

9. Fixed Wi-Fi bug in WF_CAGetChannelList() and WF_CAGetSecurity that caused failure.

10. Fixed Wi-Fi EasyConfig bug that required development boards to be manually reset even after new network was selected.

11. Fixed MRF24WB0 bug that caused assert if invalid WPA/WPA2 key was entered.

12. Fixed Wi-Fi power management bit behavior in PS-Poll frame that was causing some AP's to never send data or disconnect when in power save mode.

13. Fixed a TCP bug in which attempting to open a client TCP socket to a remote server, specified by IP address (not DNS address), that was offline, but who's MAC address was already cached by the ARP client, would result in endless back-offs. For example, when attempting to contact the remote node (that was not responding), the TCP module would have transmitted a SYN at time T=0 seconds, T=1s, T=3s, T=7s, T=15s, T=31s, T=63s, T=127s, T=255s,

etc. The exponential back-off between retransmissions would grow indefinitely until the retransmission interval would have grown so large that effectively no-retransmissions would be occurring. Assuming the application wasn't written with its own timeout to prevent endless waiting, this would prevent the socket from automatically establishing the connection to the remote server once the server came back online. With this TCP fix, the exponential back off now saturates after TCP_MAX_RETRIES (5) back offs and continues to retransmit using the same interval. By default, this means SYN transmissions will occur at T=0 seconds, T=1s, T=3s, T=7s, T=15s, T=31s, T=63s, T=95s, T=127s, etc. After 5 back-offs the retransmission interval stops growing and stays constant at 32 seconds.

14. Fixed an RSA computation bug that would cause the RSA module to never complete if you attempted to compute $y = x^e \% n$ where $e = 3$ (or similar number < 256 with only 0, 1, or 2 bits set). Thanks go to Kevin Maidment for pointing this error out and suggesting a solution. Note, that this fix to RSA.c is not distributed with the ordinary TCP/IP Stack due to United States export restrictions. To get this fix, you must repurchase SW300052. This fix is included in SW300052 v2.6 or later. If you don't have CD media to identify the SW300052 version that you have, you can test the RSA.c file that you have. RSA.c in SW300052 v2.6 has a CRC32 checksum of 0x91F66711. RSA.c in v2.5 and prior had a checksum of 0xB1E8B0CC.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenabling its DHCP server.
3. LCDBlocking.c timing for the LCD_E_IO enable signal is too fast to meet published data sheet limits for various LCD controllers when compiling for a PIC32 running at > 43MHz. Despite this potential timing violation, the LCD does normally work correctly on the precompiled PIC32 demos for the Explorer 16 at 80MHz.

v5.25 07 May 2010

Changes:

1. Added support for the Microchip MRF24WB0 802.11 WiFi controller (module part number MRF24WB0MA). This product is intended as a backwards compatible, drop in replacement to the ZG2100. The MRF24WB0MA should work with previous TCP/IP Stack versions as if it were a ZG2100M, but when the MRF24WB0MA is used with this (5.25) and future TCP/IP Stack versions, feature improvements inside the MRF24WB0 allow the TCP/IP Stack code/RAM size to be smaller and run faster.
2. Dropped support for the ZeroG ZG2100 802.11 WiFi controller. Applications that must stay with this device should continue to use TCP/IP Stack version 5.20b or earlier. All new projects or preexisting projects undergoing updates should be developed with the MRF24WB0 instead.
3. The WiFi connection management state machines now run on the MRF24WB0 instead of the PIC host, freeing up code and data space. Connection profiles can be created and the connection algorithm fine-tuned by the PIC application. In the WiFi demos see the WF_Connect function in MainDemo.c for an example of how to configure and then establish a WiFi connection. The programming model has changed to an API model which is documented in 'TCPIP Stack Help.chm'.
4. Changed "VERSION" macro definition in TCPIP.h to "TCPIP_STACK_VERSION". "VERSION" is overly generic and will likely conflict with other identical tokens one may use in their application code or source libraries.
5. Added support for the PIC24FJ256GA110, PIC24FJ256GB110 and PIC24FJ256GB210 PIMs for the Explorer 16. Note that when using the PIC24FJ256GA110 general purpose PIM, the Ethernet PICtail Plus, Fast 100Mbps Ethernet PICtail Plus, MRF24WB0MA Wi-Fi PICtail Plus, or other SPI PICtail daughter board should be installed in the middle SPI2 slot of the Explorer 16, not the ordinary topmost SPI1 slot used by other PIMs, including the PIC24FJ256GB110 and PIC24FJ256GB210 ones. The software is set up to use SPI2 for the PIC24FJ256GA110 PIM to avoid incompatibilities with silicon revision A3, which does not allow the SCK1 pin to be configured as a PPS output.
6. Added support for the PIC24FJ256DA210 Development Board.
7. Added TFTPUploadRAMFileToHost(), TFTPUploadFragmentedRAMFileToHost() and

TFTPGetUploadStatus() APIs to the TFTPc.c file. These APIs provide a very simple means of uploading a whole file to a remote TFTP server without requiring a great deal of application state machine logic. These APIs require the DNS client module to be enabled (STACK_USE_DNS must be defined, in addition to STACK_USE_TFTP_CLIENT).

8. Added a dummy DNS Server module. This server always sends the local IP address in response to all queries received. When using the PIC DHCP server, its purpose is to allow a user to type anything into a web browser (ex: http://asdf/) and still receive the web page provided by the PIC, much as a hotel or airport WiFi router will serve to you before you've paid or agreed to the network's terms of service. This DNS server module is implemented in DNSs.c, requires one UDP socket, and is enabled via the STACK_USE_DNS_SERVER option in TCPIPConfig.h.

9. Changed SPIFlash.h file defaults to target an SST SPI Flash with 4096 byte sectors instead of a Spansion Flash with 65536 byte sectors. These new defaults are, among other reasons, in support of the PIC24FJ256DA210 Development Board, which has an SST SST25VF016B on it.

10. Made TCP Keep-Alive packets consistently get sent TCP_KEEP_ALIVE_TIMEOUT (default 10 seconds) after the last socket TX or RX activity. In earlier stack versions, if the local node transmitted some data and then let the socket go idle, the first Keep-Alive packet sent would use the TCP_START_TIMEOUT_VAL (default 1 second) timer value before getting sent.

While benign in terms of application behavior, these faster than normal keep-alive transmissions were distracting when viewed in Wireshark or other packet capture tools.

11. Disabled STACK_USE_DYNAMICDNS_CLIENT option in TCPIPConfig.h by default for the TCPIP Demo App and TCPIP ENC24J600 Demo App projects. This option was enabled by default in earlier stack releases. This was done to save code size and allow out-of-box compilation on devices with 128KB of Flash when not using compiler optimizations. The TCPIP PIC32 ETH Demo App project continues to have this option enabled by default.

12. Added SNMP v2 TRAP PDU format. Macro SNMP_STACK_USE_V2_TRAP is used to enable the SNMP v2 trap format. New API function SNMPV2TrapDemo() is included to support more than one variable binding to the SNMPv2 TRAP. This API can be used for a single SNMPv2 TRAP variable varbind and is part of CustomSNMPApp.c. A multiple variable binding demo can be enabled MainDemo.c. One should not enable both SNMPTrapDemo and SNMPV2TrapDemo simultaneously. Global flag "gSetTrapSendFlag" is used to indicate the

start and end of SNMPv2 trap varbinds. If gSetTrapSendFlag is FALSE, then very next variable varbind for the SNMPv2 TRAP, is the last or only one variable varbind. If gSetTrapSendFlag is TRUE, then there is another variable varbind available to be part of the SNMPv2 TRAP PDU.

13. Added support for PIC32MX6XX/7XX external PHY's: SMSC 8700LAN and National DP83640.

14. Added schematics and BOM for the PIC32 Ethernet Starter Kit.

15. Added the Google PowerMeter demo project. Consult the "Reference Implementation for Google PowerMeter.chm" help file for more information.

16. Modified the SSL and TCP modules to create the TCPStartSSLClientEx function. This function will enable the SSL module to store supplementary data (currently only SSL Certificate Public Keys) in a structure.

17. Moved the HTTP_PORT, HTTPS_PORT, HTTP_MAX_DATA_LEN, and HTTP_MIN_CALLBACK_FREE macros from HTTP2.c to TCPIPConfig.h.

Fixes:

1. The SPIFlashEraseSector() function in the SPIFlash.c file incorrectly erased the sector specified by the current write pointer (set by calling SPIFlashBeginWrite()) instead of the specified dwAddr parameter address. This error had no impact on any TCP/IP Stack code as these parameters always matched. However, application code using the API would have been affected. Thanks go to Marc Boon for reporting this issue on the Microchip Ethernet forum.
2. Fixed ENC424J600/624J600 driver for PSP modes 2, 4, 6, and 10. The PIC's PMP PMMODE<9:8> bits were not set correctly.
3. Removed from lingering references to TickGetDiff() in FTP.c, TFTPc.c and UARTConfig.c.
4. Fixed DNS client module from returning the DNS server IP address if the DNS query failed due to a server error (i.e. DNS did respond, but did not return any records, such as when attempting to resolve a name that isn't in the DNS). DNSIsResolved() will now return 0.0.0.0 on any non-recoverable DNS error or timeout.
5. Fixed HTTP2 MPFS upload page being accessible from URLs that weren't an exact match. For example, in 5.20 and earlier, accessing `http://mchpboard/mpfsuploadASDF` would still have opened the mpfsupload page. Thanks go to Andrea Rivolta on the Microchip Ethernet Forum for identifying this error.
6. Improved UDP TX checksum code for the special case when the computed

checksum was 0x0000. According to the UDP RFC, for this corner case, the checksum should be converted to 0xFFFF before transmission to differentiate from the checksum disabled case, improving error detection by a minuscule amount.

7. Fixed GetCLKOUT() function in ENC24J600.c driver file. Previously, 0x00 would always be returned, regardless of the value in the COCON bits of ECON2. The function documentation for SetCLKOUT() and GetCLKOUT() was also corrected (had obsolete information ported over from ENC28J60 driver file).

8. Fixed DHCP client rebinding bug in which the DHCP client would request the wrong IP address if an unrelated DHCP OFFER or ACK message were received after we transmitted a DHCP REQUEST but before we received our DHCP ACK. Under rare conditions, this would have resulted in the TCP/IP stack reverting to the static or AutoIP assigned address for a few seconds between DHCP lease renewals.

9. Fixed TFTP Internet Bootloader bug in which uncommon .hex files containing a certain data pattern could not be uploaded correctly to the PIC18F97J60 family device. For these problem .hex files, a block of 32 program words (64 bytes) would remain unprogrammed (left as 0xFFFF) due to a parsing error in the bootloader's DecodeHex() function. The TFTP upload operation would succeed without reporting a programming error. The problem can be detected by using an ICD3 or similar ICSP programmer and reading the program Flash out of a device that is programmed with the bootloader and application .hex files. Compare the resulting memory dump to a device programmed only with the application .hex file. If you have devices deployed in the field with the previous bootloader and happen to generate a problem application .hex file, you can potentially work around the bootloader bug by opening the application .hex file with Notepad and appending dummy address records to the beginning to move the data around in the file. For example, at the very top of the .hex file, add lines containing ":020000040000FA" until the bootload process works correctly. You may alternatively try adding spaces at the end of any line, although this may make the .hex file incompatible with some programming utilities. Thanks go to Jonathan Seidmann for identifying and reporting this bug.

10. Fixed SNMPv2 TRAP format issue where SNMP browser was displaying all the SNMPv2 traps as SNMP version 1. SNMP v2 TRAP pdu format is rectified. Macro SNMP_STACK_USE_V2_TRAP is used to form and send a SNMPv2 TRAP PDU. SNMPTrapDemo API is used for both SNMPv1 and SNMPv2 single variable varbind trap.

11. Fixed an HTTP2.c server module initialization bug when using the PIC32MX7XX/6XX series internal Ethernet module. During initialization the HTTPLoadConn() function would overwrite over 100 bytes of PIC RAM past the end of the reserved memory allocated for the HTTP2 module. This problem would manifest itself by locking up the TCPIP PIC32 ETH Demo App-C32 demo shortly after power up if you compiled TCP/IP Stack version 5.20 with the MPLAB C Compiler for PIC32 MCUs (C32) version 1.11.
12. Fixed SSL client from incorrectly parsing for the server's public key in rare cases where the RSA Public Key Algorithm identifier was received, but the key hadn't been received by TCP yet. Thanks go to Kevin Maimdnet for identifying this error in SSL.c and reporting it via <http://support.microchip.com/>.
13. Fixed Tick.c TickGet(), TickGetDiv256() and TickGetDiv64K() APIs sometimes returning the wrong value on PIC32 platforms. On the PIC32MX3XX/4XX family devices a wrong return result would sometimes occur if using -O3 compiler optimizations (maximum speed) with the Microchip MPLAB C Compiler for PIC32 MCUs (C32). On the PIC32MX5XX/6XX/7XX family devices, such as the PIC32MX795F512L device used on the PIC32 Ethernet Starter Kit, wrong values could be returned, regardless of the compiler optimization level.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.
3. HI-TECH PICC-18 compilers are not supported in this release. The supplied HI-TECH PICC-18 MPLAB projects usually will not compile and/or link.
4. LCDBlocking.c timing for the LCD_E_IO enable signal is too fast to meet published data sheet limits for various LCD controllers when compiling for a PIC32 running at > 43MHz. Despite this potential timing violation, the LCD does normally work correctly on the precompiled PIC32 demos for the Explorer 16 at 80MHz.

v5.20 18 November 2009

Changes:

1. Added PIC32MX7XX/6XX Family integrated Ethernet controller support. The "TCPIP PIC32 ETH Demo App" folder was added to compile for the PIC32 Ethernet Starter Kit. Ethernet driver files "ETHPIC32ExtPhy.c" and "ETHPIC32IntMac.c" were added, in addition to the "ETHPIC32ExtPhyDP83848.c" file, which is a specific driver file for the National DP83848 10/100 PHY.
2. Added RFC 3927 Auto IP module. This module will automatically assign a local IP address to the node in the 169.254.xxx.xxx private address range (subnet mask 255.255.0.0) if a DHCP server is not present on the network or the DHCP client is disabled. The exact IP address chosen will be pseudo-random, but as required by the protocol, it will perform gratuitous ARPs to avoid clobbering anyone else's IP address. Also, unless there is an address collision with a preexisting node on the network, the IP address generated by the Auto IP module will not change between power cycle events (random number generator is seeded by local MAC address). To enable this module, `STACK_USE_AUTO_IP` must be defined in `TCPIPConfig.h`. When compiled in, the module defaults to enabled, but will automatically yield to the DHCP client module, which has higher priority.
3. Added "TCPIP MDD Demo App" beta application projects. Projects in this folder store the HTTP2 web pages in external FAT16/FAT32 formatted SD card or USB Mass Storage media instead of an MPFS2 formatted EEPROM or SPI Flash. For more information on these projects, see the "Running the TCPIP MDD Demo App (Beta Release).pdf" file in the Microchip\Help folder.
4. Expanded `XEEReadArray()` API's third length parameter from a `BYTE` to a `WORD`.
5. Converted all variable declarations and type casts of `TICK` data type to `DWORD`. The `TICK` typedef is now deprecated and will be removed in a future stack release. This data type conflicts with the `TICK` structure used in certain other Microchip software libraries.
6. Added `TCP_WINDOW_UPDATE_TIMEOUT_VAL` option to the `TCP.c` file (default 200ms). This timeout controls the time after calling `TCPGet()` or `TCPGetArray()` before the stack will transmit a RX window update to the remote node. Historically, the `TCP_AUTO_TRANSMIT_TIMEOUT_VAL` value was used for this purpose (default 40ms). This change decreases the net window update transmission overhead. If this adversely affects your application

RX performance (unlikely, but possible for certain communications patterns), set TCP_WINDOW_UPDATE_TIMEOUT_VAL equal to or shorter than TCP_AUTO_TRANSMIT_TIMEOUT_VAL to get the same or better behavior relative to previous stack versions.

7. Split TCP_MAX_SEG_SIZE configuration constant in TCP.c into separate TCP_MAX_SEG_SIZE_TX and TCP_MAX_SEG_SIZE_RX configuration constants. Previously, TCP_MAX_SEG_SIZE was used to limit both the maximum size of transmit and receive packets. In cases where large TX FIFOs are allocated, and the remote node advertises a large Maximum Segment Size TCP option, this change improves TCP transmit performance by roughly 10%.

8. Renamed "Internet Radio App", "Internet Bootloader App" and "WiFi Iperf App" folders to "TCPIP Internet Radio App", "TCPIP Internet Bootloader App" and "TCPIP WiFi Iperf App" respectively. These new names ensure consistent folder placement when viewing the Microchip Solutions folder with other Microchip Application Libraries installed.

Fixes:

1. Fixed SSL functionality (ex: HTTPS server) from failing when using the ENC424J600 and ENC624J600 Ethernet controllers. In stack versions 5.00 and 5.10, the BFSReg() and BFCReg() functions were being incorrectly used to set and clear CRYPTEN (EIR<15>). ENC424J600/624J600 silicon errata #6 on production silicon revision A2 prevents BFSReg() and BFCReg() from being able to modify CRYPTEN. This resulted in the SSL RSA encrypt/decrypt operations from ever finishing. The ENC424J600/624J600 errata #6 workaround is now implemented in the ToggleCRYPTEN() function in ENCX24J600.c.
2. Fixed an RSA padding error in the ENCX24J600.c's version of RSASetData() and RSASetE() functions. This fixes the Bad Record MAC problem when using SSL client APIs with the ENC424J600 and ENC624J600, as mentioned in the 5.10 stack release notes' Known Problems section. Although unknown at the time of release this problem also occurred in stack version 5.00.
3. Fixed DNS client from mishandling DNS responses that did not use name compression. Thanks go to Will Stone on the Microchip Ethernet forum for identifying this bug.
4. Fixed an ExtractURLFields() API bug which would incorrectly parse the URLs containing other URLs. Ex:
"http://www.google.com/search?q=http://www.microchip.com/"
5. Fixed TickGet(), TickGetDiv256(), and TickGetDiv64K() APIs from potentially returning an incorrect time value (0x10000 ticks less than it should have)

on rare occasions when using a PIC32 and with compiler optimizations turned on. The Tick.c module was also revised so that the IEC0 register does not get written to via a load-modify-store operation on PIC32s so that it is now possible for other application ISR functions to write to IEC0 without risking state corruption.

6. Fixed PIC32 Starter Kit Debugger losing access to the PIC32 target when the project was run. JTAG was being disabled at run time, but the PIC32 Starter Kit Debugger requires JTAG to communicate with the debug executive. JTAG is now conditionally disabled on PIC32s when the `__MPLAB_DEBUGGER_PIC32MXSK` macro is undefined.

7. Fixed a Berkeley sockets API bug in which calling `closesocket()` on a `SOCK_STREAM` type socket (TCP) did not actually close the socket if the remote node did not first send a FIN to the local node. This would leak a TCP socket each time the affected API calling sequence occurred and result in no FIN getting transmitted to the remote node.

8. Fixed an HTTP2 filename parsing bug that would occur when a web browser submitted a request for a file with hex encoded characters in it. For example, with stack version 5.10 and Firefox 3.5.3, typing "http://mchpboard/%70rotect" into the URL field would have resulted in an HTTP 404 not found error when "http://mchpboard/protect/index.htm" should have been returned instead. Thanks go to Steve Tuttle for reporting this issue and suggesting a solution.

9. Fixed a Berkeley sockets API bug in which calling `recvfrom()` on a datagram type socket (UDP) would return an incorrect remote IP address and port number when the from pointer was non-NULL.

10. Fixed HTTP2 server bug in which the `HTTPReadPostName()` function was failing to convert the field name from URL encoding to plain-text. If the browser posted, for example, a field named "Stock Remaining", it would have been incorrectly returned from `HTTPReadPostName()` as "Stock+Remaining".

11. In Stack 5.10, any new values you saved into AppConfig via the Network Configuration demo web page would have been mishandled for WiFi projects. `HTTPPostConfig()` in `CustomHTTPApp.c` of the TCPIP WiFi Demo App and TCPIP lperf Demo App projects were corrected so that they now write a magic 0x61 marker into EEPROM/SPI Flash address 0x0000 to indicate that the AppConfig structure is valid in EEPROM/SPI Flash. This prevents the `InitAppConfig()` function in `MainDemo.c` from restoring the default settings when changing the values through the Network Configuration page.

12. For WiFi projects, a Gratuitous ARP Work-around was implemented to work

around cases where access points send broadcast messages at data rates that the ZG2100 cannot listen to. The define `USE_GRATUITOUS_ARP` (in `TCPIPConfig.h`) turns this feature on or off.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.
3. HI-TECH PICC-18 compilers are not supported in this release. The supplied HI-TECH PICC-18 MPLAB projects usually will not compile and/or link.
4. ENC624J600 PSP modes 2, 4, 6, and 10 do not work at this time. Some Parallel Bit Bang modes may not work either. Some minor firmware changes are needed.
5. TCPIP ENC624J600 Demo App-C18.mcp project does not compile by default using MPLAB C Compiler for PIC18 MCUs (C18) version 3.34. There is not quite enough program memory available on the PIC18F97J60 for the large number of selected stack features to allow linking. To get this project to compile, turn on compiler optimizations or disable one of the modules in `TCPIPConfig.h` (ex: comment out `STACK_USE_DYNAMICDNS_CLIENT`).

v5.10 29 July 2009

Changes:

1. Added SSL capability to the Telnet server. If `STACK_USE_SSL_SERVER` is defined, the Telnet server will now listen on port 992 for SSL secured connections. If you do not have a telnet client, you can use an SSL proxy, such as stunnel (<http://www.stunnel.org/>) to add SSL/TLS support to any telnet client.
2. Moved a number of string pointer tables in the HTTP.c, HTTP2.c, FTP.c, and DynDNS.c files to allocate in ROM instead of RAM. This reduces around 120

bytes of RAM usage in the HTTP2 server when compiled for the PIC18 or PIC24/dsPIC platforms. The gains are even greater on PIC32 platforms.

3. Added redefinition of SPIRAM*(), SPIFlash*(), and XEEPROM*() functions so that when compiled and used without proper HardwareProfile.h definitions, a more descriptive linker error will be generated instead of a mysterious symbol not found error.

4. Added several new APIs:

- ExtractURLFields() in Helpers.c. This function provides an easy means of parsing a URL string and extracting the protocol, hostname, port, file path, etc. Currently, this function is commented out to save code space as no stack modules require it. However, it should work correctly if you simply uncomment it (remove the #if 0...#endif around it).
- strnchr() in Helpers.c. Finds the first occurrence of a character within a string but limited to a maximum length before giving up.
- TCPPeek() and TCPPeekArray() in TCP.c. Reads from a TCP socket's RX FIFO buffer without removing the data from the stream.
- TCPClose() in TCP.c. Disconnects a socket from the remote node (if connected) and closes the socket handle, including for server type sockets. This function is identical to the TCPDisconnect() API except for the handling of server sockets. TCPDisconnect() returns server sockets to the listening state and maintains the socket handle. TCPClose() closes the socket and frees all associated resources, invalidating the socket handle.

5. Updated the DHCP client module:

- Modified so that it wouldn't attempt to transmit DHCP Discover packets when the MAC layer reports no link (MACIsLinked() == FALSE). This avoids main() while(1) loop performance degradation when you unplug the Ethernet cable or lose association to your access point.
- Added capability of performing DHCP discovers and requests without setting the BOOTP broadcast flag. Now, the DHCP client module will start up and attempt to obtain an IP address with the broadcast flag set, but if it fails the next DHCP retry will attempt to obtain the IP address with the broadcast flag cleared. The flag will toggle back and forth between unicast mode and broadcast mode if no DHCP server responds. This feature improves compatibility with certain DHCP servers and WiFi access points.
- Added several new APIs including DHCPInit(), DHCPIsEnabled(),

DHCPStateChanged(), DHCPIsBound(), and DHCPIsServerDetected().

- Removed the DHCPFlags DHCP_CLIENT_FLAGS global variable. Use the above named APIs now to get equivalent functionality.

- Removed the DHCPBindCount global variable. To detect if the DHCP state has changed, poll the new DHCPStateChanged() function.

- Removed the DHCPReset() API. To perform this operation, now call the DHCPInit() API. Use 0x00 for the vInterface parameter.

6. Removed deprecated TickGetDiff() macro. To get a tick difference, just subtract the two values in-line. This macro was removed because it promoted confusing code. Ex: a-b is different from b-a. However, it was not contextually obvious which of the two was returned when TickGetDiff(a, b) was called.

7. Added PIC32MX460F512L USB and dsPIC33FJ256GP710 PIM support to the Explorer 16 hardware profile for the TCPIP WiFi Demo App and WiFi IPerf App projects.

8. Added all files needed for SSL (assuming the crypto libraries are present) to the TCPIP WiFi Demo App-C30 and TCPIP WiFi Demo App-C32 projects.

9. Converted TCPIP Demo App, TCPIP WebVend App, Internet Radio App, and Internet Bootloader App MPLAB Build Directory Policy to compile in the project folder instead of the source folder. This reduces the dependencies on the MPLAB project include path and allows new projects to be created by copying one of the pre-existing folders (ex: copy "TCPIP Demo App" to "My App") without having problems including the wrong HardwareProfile.h and TCPIPConfig.h files.

10. Changed EEPROM/SPI Flash AppConfig record valid flag from 0x60 to 0x61 in the TCPIP WiFi Demo App and WiFi Iperf App projects. This will force the various EEPROM settings to get erased when switching between Ethernet and WiFi projects. This is done since the AppConfig structure changes when using WiFi (SSID string is added).

11. The Wifi Iperf App and TCPIP WiFi Demo App projects have been optimized for better performance.

Fixes:

1. Fixed a TCPDisconnect() API bug in which the last few bytes of data (up to the TCP socket's TX FIFO size less 532 bytes) was not transmitted and no FIN was sent out if the TX FIFO was full of data when TCPDisconnect() was called. This problem could have only occurred for TCP sockets with a large TX FIFO (≥ 532 bytes). This problem could have been observed in stack version 5.00's "TCPIP Demo App-C32 EXPLORER_16 32MX360F512L ENC624J600

PSP 9.hex" precompiled application, among others, if you connected to the TCPServerTest.c module and then attempted to simultaneously access the web server. The web server was returning data very slowly and failing to send the last parts of each file requested by the browser.

2. Eliminated a potential buffer overflow vulnerability from the HTTPHeaderParseContentLength() function in HTTP2.c. If an oversized or malformed Content-Length header is sent from the web client, the function will now gracefully fail by returning an HTTP 400 Bad Request error page.

Thanks go to Mark Philipp for identifying this error and suggesting a solution.

3. Fixed a TCPOpen() problem in which the stack would continuously flood the network with nearly back-to-back ARP query packets if a client socket was created that specified a non-reachable remote IP address (ex: local gateway was offline, or for destinations on the same subnet, the actual remote node was offline). This problem would occur only after a few minutes (<10) had passed since the PIC was last reset. Thanks go to Sergey of DPS TELECOM for reporting this problem.

4. Fixed linking problem with BigInt_helpers.S (PIC24/dsPIC only) when targeting a PIC with more than 8KB of RAM. The interface registers (_iA, _xA, _iB, _xB, _iR, and _wC) are now forced into near RAM.

5. Cleaned up some uninitialized variable warnings in SNMP.c.

6. Fixed a sequence variable traversal bug in SNMP.c.

7. Cleaned up a large number of unsigned integer to signed integer comparison warnings produced by the MPLAB C Compiler for PIC18 MCUs (C18) version 3.32. With earlier versions of this compiler, these warnings would only be generated as messages, so they did not get displayed by default.

8. Some ENCX24J600 parallel bit bang modes work now. PSP Mode 5 indirect has been tested.

9. SSL client and server capabilities now work when using the ZeroG ZG2100M WiFi interface. In the 5.00 stack release, attempting to enable the STACK_USE_SSL_CLIENT or STACK_USE_SSL_SERVER TCPIPConfig.h options with this network controller would have resulted in an error trap. If an LCD was present, the LCD would display "encRdPtrRAWId = encWrPtrRAWId" when the error occurred.

10. The WiFi Iperf App demo locked up when an invalid command was entered at the serial port console. This is now fixed.

11. The WiFi Iperf App demo locked up when running with a PIC32 if iwconfig was typed at the serial port console. This is now fixed.

12. The Wifi Iperf App demo, when running on the PIC24 and PIC32, and compiled with the `-Os` option (min code size optimization), did not work. This is now fixed.

13. Change a lot of BerkeleyAPI.c internals. This may fix a number of BSD API problems.

14. Fix a problem with SNMP variables being inaccessible with certain unique PEN numbers.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.
3. HI-TECH PICC-18 compilers are not supported in this release. The supplied HI-TECH PICC-18 MPLAB projects usually will not compile and/or link.
4. ENC624J600 PSP modes 2, 4, 6, and 10 do not work at this time. Some Parallel Bit Bang modes may not work either. Some minor firmware changes are needed.
5. SSL client code doesn't work with ENC424J600/624J600 devices. The remote server terminates the connection reporting a bad record MAC (Message Authentication Code). The SSL client does work with other controllers.

v5.00 27 April 2009

Changes:

1. Added ZeroG ZG2100 802.11 WiFi controller support. The new TCPIP WiFi Demo App and WiFi Iperf App projects have been added, which default to using this controller.
2. Added Microchip ENC424J600/624J600 10/100 Ethernet controller support. Support for this controller is provided by the new ENC424J600.c/.h files which perform the same role as the ENC28J60.c/.h or ETH97J60.c/.h files.

Precompiled .hex files for the ENC624J600 controller require the use of the new Fast 100Mbps Ethernet PICtail Plus daughter card (AC164132). This product is not available at the time of the 5.00 TCP/IP stack release.

However, it is anticipated to be available for purchase on www.microchipdirect.com in CQ3 2009.

3. Significantly updated the Internet Radio App project. Previously, radio stations were hard coded into program memory at compile time. Now, a dynamic Shoutcast directory client has been implemented which allows retrieval of radio stations at run time, offering endless stations you can tune into. The web pages for the radio have also been updated to allow control and status reporting of the board from a web browser.

4. Update SNMP Server (Agent) module to support SNMPv2C. The default Demo App web pages now include an SNMP reconfiguration capability to set the read and write community strings.

5. Added ICMPSendPingToHost() and ICMPSendPingToHostROM() APIs to ICMP (ping) client module. These two APIs are available only when STACK_USE_ICMP_CLIENT and STACK_USE_DNS is defined in TCPIPConfig.h. These functions allow pinging of DNS hostnames directly without the need for the application to convert the hostname to an IP address first by manually calling the DNS client APIs. With this addition, the PingDemo.c file was updated to ping the hostname "ww1.microchip.com" instead of a static IP address. Previously, the PingDemo would stop working a couple of months after the stack was released, due to the IP address of the www.microchip.com server dynamically changing. If the DNS module is not enabled, the ping demo will instead ping the local gateway IP address instead of ww1.microchip.com.

6. Updated TCPPerformanceTest.c code. The previous version would generate incorrect speed calculations at high data rates (ex: >1Mbyte/sec).

7. Added multiple connection support to Telnet server example module. To allow multiple connections, define MAX_SIMULTANEOUS_CONNECTIONS in Telnet.c greater than 1 and create an equal number of TCP_PURPOSE_TELNET type TCP sockets in the TCPSocketInitializer[] definition in TCPIPConfig.h.

8. Added more randomness to the local port selection when opening a client-mode TCP socket. This reduces the risk of reusing a previously used port number if the user power cycles the device.

9. Updated XEE* SPI EEPROM API functions. Writes are no longer required to start on an EEPROM page boundary, and writes can now be arbitrarily long without having to call XEEEndWrite() at each page boundary. Additionally, the XEEWriteArray() API has been added, which performs a similar operation

to the SPIFlashWriteArray() API (but with no special erase cases to worry about).

10. Decoupled AppConfig storage in external SPI EEPROM or SPI Flash option from MPFS_USE_EEPROM and MPFS_USE_SPI_FLASH options. MainDemo.c will now save the AppConfig structure in external non-volatile memory, even if MPFS is unused (no HTTP or SNMP server modules enabled) or MPFS is using internal Flash program memory to store web pages/bib information. This change also allows the XEE*() and SPIFlash*() non-volatile read/write functions to be available at all times (even if MPFS is unused), as long as the appropriate hardware pinout definitions are present in HardwareProfile.h. SPI Flash and SPI EEPROM are no longer mutually exclusive with each other. However, if both are enabled simultaneously, AppConfig will be stored in the EEPROM, not the SPI Flash.

11. Added required SSL files to TCPIP Demo App MPLAB projects. SSL capabilities can now be turned on directly via the STACK_USE_SSL_SERVER and STACK_USE_SSL_CLIENT options in TCPIPConfig.h for these projects, assuming appropriate crypto libraries are installed (SW300052 available from <https://www.microchipdirect.com/>). With this change, the historical "SSL Demo App" folder has been removed.

13. Updated HardwareProfile.h files. This includes the addition of PIC18 Explorer board support, removal of the PICDEM Z profile, changes to the HI-TECH PICC-18 profiles for newer compilers, among other changes.

14. Added a TCP and UDP performance test measurements table to TCPIP Stack Help (TCPIP Stack Help.chm). Access this from the "Microchip TCP/IP Stack" book, "Stack Performance" page.

15. Updated MPFSlib project (Microchip.MPFS.dll file) so that C18 and C32 output from the MPFS2.exe utility is now identical for MPFS2 images. The generated .c file is now compatible with both C18 and C32 compilers simultaneously. Previously, the images generated for C18 would compile successfully for C32 projects, but would potentially operate incorrectly when compiler optimizations were turned on. Images generated for C32 would compile successfully and work on C18 projects, but the C18 compiler would take a very long time to process the file each time you rebuilt your MPLAB project. Now, the image generated for C18 matches the image generated for C32 and it will compile fast and work correctly on both platforms, even with compiler optimizations turned on.

16. Added schematics and BOMs for the Ethernet PICtail, Ethernet PICtail Plus, Fast 100Mbps Ethernet PICtail Plus, Internet Radio, PICDEM.net 2, and ZeroG

ZG2100M PICtail development boards to the "Microchip\TCPIP Stack\Demo Board Files" folder.

Fixes:

1. Fixed a denial of service vulnerability in the `NBNSGetName()` function of the `NBNS.c` file. Previously, if a deliberately malformed packet was received, the PIC RAM could have become corrupted. Thanks go to David Talmage for finding this vulnerability.
2. Fixed Timer1 interrupt flag clearing code on PIC32 products. Previously, the `Tick.c` module was clearing the interrupt flag in an unsafe manner which could have corrupted other interrupt flags in the `IFS0` register. Thanks go to Leon van Snippenberg working on the AVIX-RT RTOS for pointing this error out on the Microchip forums.
3. Fixed SNMP up-time variable. Previously the `CustomSNMPApp.c` module would respond with the number of Tick API ticks that elapsed, not the number of 10ms time slices that elapsed. The SNMP standard uses 10ms as its time base.
4. Fixed `BigInt_helper.asm`'s `_masBI()` and `_masBIROM()` functions when the `Br` parameter's length modulo 4 was equal to 1 or 2. This bug previously caused the `BigIntMod()` function to sometimes go into an endless calculation loop on PIC18 products when using the SSL libraries and certain combinations of modulus data and length were used. Thanks go to Vasil Stoianov on the Microchip Ethernet forum for running into this defect and reporting it.
5. Fixed `SSLSessionNew()` so that it wouldn't "lose" SSL sessions after waiting a few hours. This would previously make it impossible to make new SSL connections after a while, but then after a few more hours, the sessions would become free again. Thanks go to Jim Stephens for identifying this issue and finding the solution.
6. Fixed an SSL 2.0 antique client hello record length calculation bug occurring when a received record was > 255 bytes.
7. Added retransmission capability to `SendNotification()` function in `CustomSNMPApp.c`. Previously, if an SNMP trap were sent, but the initial ARP query or response was lost on the network, the `SendNotification()` code would have deadlocked, and suppressed all future transmission of SNMP traps.
8. Fixed DNS client timeout if the DNS server is unable to be ARPed. Previously, the DNS client would retry ARPing the DNS server indefinitely if it was offline. Now, the DNS client will correctly abort if too many attempts to ARP the DNS server fail. Thanks go to Phil "andersop" on the Microchip Ethernet forum for identifying this error.

9. Suppressed transmission of a TCP RST packet to an unknown IP or MAC address if the TCPDisconnect() function was called on a client mode socket that was not finished with ARP or DNS resolution yet. Thanks go to Phil "andersop" on the Microchip Ethernet forum for pointing this behavior out.
10. Fixed TCP socket from disconnecting if the remote receive window was zero and TCPFlush() was still called. Thanks go to Bob Topper for identifying this issue and suggesting a solution.
11. Fixed Tick.c module returning incorrect values when TickGet() or other API was used with compiler optimizations turned on. Wrong values were observed when using MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs version 3.12.
12. Fixed a number of SPI communications problems that could occur when compiler optimizations were turned on. The ENC28J60 was observed to not work correctly on the dsPIC33FJ256GP710 processor when compiled with MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs version 3.12.
13. Fixed possible MPFS2 error when using an ASM30 .s image where MPFS_Start would be read using the wrong PSVPAG setting. You must rebuild your MPFS2 image file (ex: MPFSImg2.s) with this stack version's MPFS2.exe utility to get this correction applied.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenabling its DHCP server.
3. HI-TECH PICC-18 compilers are not supported in this release. The supplied HI-TECH PICC-18 MPLAB projects usually will not compile and/or link.
4. ENC624J600 PSP modes 2, 4, 6, and 10 do not work at this time. Parallel Bit Bang mode does not work either. Some minor firmware changes are needed.

v4.55 10 November 2008

SSL Note:

RSA.c and ARCFOUR.c have not changed between the 4.50 and 4.55 releases.

Although the precompiled SSL Demo App .hex files will differ, you can continue to use the previous TCP/IP Stack v4.50 Encryption Add-on with this 4.55 stack version.

Changes:

1. Added DNS client support for a secondary DNS server address. Previously, the AppConfig.SecondaryDNSServer setting was unused. Now, the DNS client module will automatically swap the AppConfig.PrimaryDNSServer and AppConfig.SecondaryDNSServer values after any DNS query timeout (or ARP timeout for the DNS server) and attempt the query with the alternative server. If AppConfig.SecondaryDNSServer is disabled by setting it to the IP address 0.0.0.0, the DNS client will only use the AppConfig.PrimaryDNSServer value and never swap the values. With this change, the DHCP client was also updated. If the DHCP server does not specify a secondary DNS server, then the DHCP client will now set the AppConfig.SecondaryDNSServer value to 0.0.0.0. Previously, it would change the AppConfig.SecondaryDNSServer setting only if the remote DHCP server offered a secondary DNS server.

Fixes:

1. Updated Internet Bootloader App project to correctly detect if the configuration bits are being changed or not. Previously, the bootloader always thought the configuration bits were being changed and thus had to always erase the last Flash page (largest memory address) twice for each firmware update. This did not cause any functional problems or specification violations, but it would decrease the effective Flash endurance of the last page.
2. Fixed a TCP socket memory corruption bug that would occur if TCPGetRemoteInfo() API was called twice with different socket handles without an intermediate call to any other TCP API that takes a TCP_SOCKET input. Thanks go to Bob Topper for identifying this problem and suggesting a solution.
3. Fixed the UDPIsGetReady() function so that it returns the number of bytes remaining in the packet based on the current read location. This is the same behavior as stack versions 4.18 and earlier. In stack versions 4.50 and 4.51, the UDPIsGetReady() function would always return the total number of bytes in the current packet, regardless of how many bytes the read

pointer had been advanced through the UDPGet() and UDPGetArray() functions.

Thanks go to Bob Topper for identifying this problem and suggesting a solution.

4. Fixed demo admin web page in TCPIP Demo App project so that the last byte of the MAC address can be changed, independent of the format it was entered by the user.

5. Fixed a buffer overflow bug that would occur when using the SSL server during hashing of the server certificate for the initial handshake. This error previously caused several bytes of random variables elsewhere in the project to get overwritten for each SSL connection.

6. BSD sockets API was updated to fix some issues.

7. LCDBlocking.c was updated to relax start up timing. This timing fix is specifically needed to support Explorer 16 boards with a Truly TSB1G7000 display (Novatek NT7603H controller).

8. Removed four uses of Arial Black font in MPFS2.exe utility. On some rare PC configurations, the use of this font caused the executable to not run.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.

2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenabling its DHCP server.

3. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC_EXPLORER + Ethernet PICtail.

4. MAC.h RXSIZE precompiler test for proper range doesn't work. This is not a functional problem, just a compile-time configuration test. Ensure that you don't over allocate TCP_ETH_RAM_SIZE or MAX_HTTP_CONNECTIONS.

5. HI-TECH PICC-18 STD 9.51PL1 cannot compile DynDNS.c. It raises an "inconsistent type" error while trying to perform a ROM pointer to integer

cast. The older 9.50PL3 compiler release is required to compile this file.

6. HI-TECH PICC-18 STD 9.50PL3 does not initialize several static variables correctly on reset. This behavior breaks many stack modules in the TCPIP Demo App and TCPIP WebVend App projects. Additionally, string printing functions do not work correctly, so the supplied

"TCPIP Demo App-HITECHPICC18 PICDEMNET2 18F97J60.hex" and "TCPIP WebVend App-HITECHPICC18 PICDEMNET2 18F97J60.hex" files may not correctly print the board's DHCP assigned IP address on the board's LCD (if present) and UART. To avoid these severe problems, use the Microchip MPLAB C Compiler for PIC18 MCUs. A free student edition can be downloaded from <http://www.microchip.com/c18>.

v4.51 24 July 2008

IMPORTANT NOTE: You must use MPLAB 8.10 or higher to successfully open the MPLAB projects.

SSL Note:

RSA.c and ARCFour.c have not changed between the 4.50 and 4.51 releases. Although the precompiled SSL Demo App .hex files will differ, you can continue to use the previous TCP/IP Stack v4.50 Encryption Add-on with this 4.51 stack version.

Changes:

None. This release includes bug fixes only. It is very important that applications using the ENC28J60 get fix item 7, below.

Fixes:

1. TCPOpen() was previously failing if you used it to start a connection with a remote hostname, but the DNS module failed to resolve the remote address on the first try. This, for example, would occur if you powered up your board and tried to connect to a remote server before the Ethernet cable was attached. Once the Ethernet cable was attached, the socket would attempt to resolve and connect to a garbage address. The Internet Radio application would sometimes not begin playing the default station upon power up because of this problem.
2. Set SEQ.ACK = 0 for outbound TCP SYN packets. This fixes a connection compatibility problem with certain paranoid TCP/IP stacks that would validate

this field even though the ACK flag was clear. This problem would previously cause the Microchip TCP/IP stack to be unable to connect client-mode TCP sockets to certain rare servers/services. Thanks go to Jean LE TOUTOUR for finding one of these problem servers.

3. MPFSOpen() and MPFSOpenROM() for MPFS2 could leak a file handle if a name hash matched but no complete file name did. This has been corrected to prevent potential DOS attacks on the HTTP2 web server. Thanks to David Tan on the Microchip Ethernet forum for identifying this issue.

4. Fixed a bug in MPFS2.1 that caused compile errors when MPFS Classic images were generated for ASM30 containing files whose length was either zero or a multiple of 12.

5. Fixed an issue in HTTPPostConfig() that caused it to ignore the flag that was set when invalid IP address input was detected. This issue only affects the example configuration page and only exists in v4.50 (prior versions functioned correctly). Also corrected an issue where user input could potentially overflow into part of the shadow AppConfig in the same function. Thanks to prin3nroll3 on the Microchip Ethernet forums for identifying both of these issues.

6. Implemented Explorer 16 development board 5V LCD errata workaround to LCDBlocking.c. This corrects the A/D converter from returning erratic readings on certain Explorer 16 boards. LCD I/O pins are now continuously driven by the microcontroller instead of going high impedance when idle.

7. Fixed a critical ENC28J60 revision B7 errata workaround problem in the ENC28J60.c, MACFlush() function. Previously, the code was checking for an EREVID register value of 0x07 for silicon revision B7. This was incorrect. Silicon revision B7 actually has an EREVID value of 0x06. Note that this problem was caused by an incorrect EREVID value published in DS80349A, the B7 silicon errata documentation. Make sure to use DS80349B or later.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.

2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If

this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.

3. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC_EXPLORER + Ethernet PICtail.

4. MAC.h RXSIZE precompiler test for proper range doesn't work. This is not a functional problem, just a compile-time configuration test. Ensure that you don't over allocate TCP_ETH_RAM_SIZE or MAX_HTTP_CONNECTIONS.

5. HI-TECH PICC-18 STD 9.51PL1 cannot compile DynDNS.c. It raises an "inconsistent type" error while trying to perform a ROM pointer to integer cast. The older 9.50PL3 compiler release is required to compile this file.

6. HI-TECH PICC-18 STD 9.50PL3 does not initialize several static variables correctly on reset. This behavior breaks many stack modules in the TCPIP Demo App and TCPIP WebVend App projects. Additionally, string printing functions do not work correctly, so the supplied "TCPIP Demo App-HITECHPICC18 PICDEMNET2 18F97J60.hex" and "TCPIP WebVend App-HITECHPICC18 PICDEMNET2 18F97J60.hex" files may not correctly print the board's DHCP assigned IP address on the board's LCD (if present) and UART. To avoid these severe problems, use the Microchip MPLAB C Compiler for PIC18 MCUs. A free student edition can be downloaded from <http://www.microchip.com/c18>.

v4.50 02 June 2008

IMPORTANT NOTE: You must use MPLAB 8.10 or higher to successfully open the MPLAB projects. Also, ensure that the latest C compiler is used. This release was tested against MPLAB C Compiler for PIC18 MCUs version 3.20, MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs version 3.10, MPLAB C Compiler for PIC32 MCUs version 1.01, and HI-TECH PICC-18 version 9.50PL3 (STD). Earlier compilers may not be able to compile this TCP/IP stack release.

Changes:

1. Added SSL 3.0 client capabilities, including SMTP over SSL. The SSL modules supports up to 1024-bit RSA handshakes and 128-bit ARCFOUR bulk encryption. This can be demonstrated using the SMTP client. SSL

server support is functional, but a key generation utility is not yet provided and support over HTTPS is not yet reliable with all browsers.

IMPORTANT: Encryption software is covered by US Export Control law, so it is not directly downloadable from the Microchip website. To use the encryption modules, you must order SW300052 from microchipDIRECT [<https://www.microchipdirect.com/>] and install the required libraries.

2. Added Berkeley Sockets Distribution (BSD) API translation layer. You can now call the well know Berkeley APIs instead of or in addition to the Microchip specific APIs. To use this new functionality, define `STACK_USE_BERKELEY_API` and configure `BSD_SOCKET_COUNT` in `TCPIPConfig.h`.

Three new source code demos are provided to demonstrate this API:

`BerkeleyTCPClientDemo.c`, `BerkeleyTCPServerDemo.c`, and `BerkeleyUDPClientDemo.c`. The TCP client demo is identical to the `GenericTCPClient.c` demo, but implemented using Berkeley Sockets. The UDP client demo is similarly identical to the `SNTP.c` client. The TCP server demo listens on TCP port 9764 and will echo any traffic received back to the sender. It allows up to 3 simultaneous connections when there are an adequate number of sockets defined in the `TCPSocketInitializer[]` array in `TCPIPConfig.h`.

3. Added support for Dynamic DNS services. See the Dynamic DNS Client module in the TCP/IP Stack Help for details. Presently, `dyndns.org`, `dyndns.com`, `no-ip.com`, and `dns-o-matic.com` are supported.

4. Added the Microchip TCP/IP Configuration Wizard to the Utilities folder, facilitating easier configuration of the TCP/IP Stack through a graphical application.

5. Restructured `TCPIPConfig.h` to remove rule-enforcement logic, placing the removed sections in `TCPIP.h`. Many other project structure changes were also made to clean up the general distribution appearance.

6. Increased DHCP Server default lease duration to 60 seconds instead of 15 seconds. Some computers were losing their IP lease before performing a renew operation with only a 15 second lease.

7. Removed `CLOCK_FREQ`, `INSTR_FREQ`, and `PERIPHERAL_FREQ` macro definitions. `GetSystemClock()`, `GetInstructionClock()`, and `GetPeripheralClock()` now return these respective values. This change was made for compatibility with other Microchip software libraries.

8. Added TCP Fast Retransmission capability. Whenever three duplicate ACK packets arrive, the stack will now immediately perform a retransmit operation. This greatly improves recovery latency whenever the network

loses a packet for applications that stream TX data using TCP.

9. Improved TCP Keep Alive mechanism to automatically close TCP sockets which do not receive any keep-alive responses for `TCP_MAX_UNACKED_KEEP_ALIVES` (default 6) times. This means that, by default, any connection that catastrophically breaks without notifying us (ex: user unplugs cable, Internet connection goes down, etc.) will time out and automatically close after 60 seconds (`TCP_MAX_UNACKED_KEEP_ALIVES * TCP_KEEP_ALIVE_TIMEOUT`). Server oriented sockets will return to the listening state. Client oriented sockets will close, but the `TCP_SOCKET` handle will continue to remain valid until the application calls `TCPDisconnect()`. Applications can check if the socket became disconnected and reset by calling `TCPWasReset()` or `TCPIsConnected()`. Note that this keep alive implementation will only close sockets that are broken (remote node is not responding to TCP requests). It will not close or otherwise interfere with idle connections in which the application is not transmitting or receiving data and wishes to keep the connection open.

10. Added a TCP RX SYN queue of depth `TCP_SYN_QUEUE_MAX_ENTRIES` (default 3).

This queue automatically saves incoming SYN packets destined for a local server port which is already connected to a different client. When the client disconnects, the SYN data is pulled out of the queue and the socket immediately attempts to connect to the next client. This improves connect time performance since the remote client no longer has to retransmit the SYN request if it was unserviceable the first time around. This is most apparent with the HTTP/HTTP2 servers which previously performed poorly with certain modern web browsers which attempt to open many simultaneous connections to the web server, such as Mozilla Firefox 3 beta 5 and Apple Safari 3.1. Entries in the queue automatically time out after `TCP_SYN_QUEUE_TIMEOUT` (default 3 seconds) so as to prevent the queue from filling up permanently if several connection requests arrive for a service that is in use and will not be available for an extended period.

11. Modified the structure of the MPFS2 FAT (now known as MPFS2.1) to include name hashes first. This speeds up opening files by 25%, and makes opening index files nearly instant.

12. Updated the MPFS2 Utility. MPFS2.1 now supports the new FAT structure and provides a cleaner interface. It also writes images to disk as they are created, which eliminates the `IndexOutOfBounds` exceptions some users had reported. Finally, uploads are now truly multi-threaded.

13. Source code to the MPFS2.exe PC utility is now released. Find it in the

Microchip Solutions\Microchip\TCPIP Stack\Utilities\Source\MPFS21 folder. This project is designed to compile with Microsoft Visual C# 2008 Express Edition.

14.Added support for SST25VFxxxB serial flash parts in 2, 4, 8, 16, and 32Mbit densities. These parts can be used to replace EEPROMs for storing MPFS images (both versions) and custom data.

15.Added HTTPReadPostName, HTTPReadPostValue, and HTTPReadPostPair functions to facilitate easier processing of data arriving via POST.

16.Split HTTPAuthenticate API into separate functions: HTTPNeedsAuth and HTTPCheckAuth. This function was already split internally, and didn't make sense as a single API.

17.Updated DHCP client to close its UDP socket when idle (bound state) to save a small amount of resources.

18.Removed LED_IO macro from all hardware profiles because it is not suitable for use on certain hardware platforms that have non-contiguous LEDs or reversed bit ordering. Use the new LED_GET() and LED_PUT(val) macros to read and write to all of the LEDs at once.

19.Added Ethernet Hash Table Calculator.exe to the Utilities folder and start menu. This tool will calculate the correct bit that you must set in the EHT0-EHT7 registers on the ENC28J60 and PIC18F97J60 family devices for using the Hash Table RX filter. This is useful only for fixed MAC addresses known at design time. For addresses that are known at run time, use the SetRXHashTableEntry() function in the ENC28J60.c or ETH97J60.c files to set the correct EHT0-EHT7 bit.

Fixes:

1. Fixed a buffer overflow data corruption issue in the FTP module that arises when too many parameters were passed on the command line.
 2. Moved TCPWasReset checking in HTTP2 to execute for every socket on every loop. Previously, it would only execute when a socket reconnected, which caused the RX buffer to not resize until after data was received. Some platforms (notably FF2 on Ubuntu) would stall if the initial advertised RX window was too small, and this change corrects that issue.
 3. Updated SendSystemReset() and MACInit() initialization routine in ENC28J60.c. Previously, if the ENC28J60 was placed into sleep mode by calling MACPowerDown(), the SendSystemReset() command would not work anymore. This would leave the ENC28J60 in power down if the host PIC was ever reset. SendSystemReset() should work for all conditions with this update.
- Thanks go to Rob Haverkort on the Microchip Ethernet forum for identifying this problem.

4. Fixed an alignment bug in HTTP2 that caused redirects to fail when the MPFS2 image was stored in Flash program memory. Thanks to Todd Boaz on the Microchip Ethernet forum for identifying this bug, and Chen Qu for posting a solution.
5. Fixed SNTP client from losing accuracy if you called `SNTPGetUTCSeconds()` 10s of thousands of times since the last server synchronization. Thanks go to "pic123" on the Microchip Ethernet forum for noticing this error.
6. Fixed a `TickGet*()` API problem where the returned tick value could be off by 64K ticks occasionally on PIC24, dsPIC30/33, and PIC32 processors. This bug was previously fixed in stack versions 4.13 and 4.16, but it was unintentionally recreated in 4.18 due to PIC32 changes.
7. Fixed UART2TCPBridge module from failing to connect to a remote server when `USE_REMOTE_TCP_SERVER` was defined.
8. Fixed an issue that prevented SNMP SETs on 16 and 32 bit parts when using MPFS2. Thanks go to Milena K on the Microchip Ethernet forum for identifying this problem.
9. Fixed a rare buffer corruption issue that could occur with UDP if TCP was also enabled.
10. Fixed a Tick rollover error in HTTP2. Thanks go to Paul Bixel on the Microchip Ethernet forum for identifying this problem.
11. Fixed an MPFS2 bug in which an excessive value to `MPFS_SEEK_REWIND` may have failed to return an error. Thanks go to Paul Bixel on the Microchip Ethernet forum for identifying this problem as well.
12. SMTP Client now sends EHLO when using authentication. Previously, the HELO command was used, even with authentication enabled. Using HELO with authentication creates incompatibilities with certain SMTP servers.
13. Improved Internet Bootloader robustness by retransmitting ACKs in response to data retransmissions by the remote sending node. Previously, if an ACK packet was lost before reaching the sending node, the TFTP upload would fail and need to be restarted. Thanks go to "coolvibe" Dave Collier on the Microchip Ethernet forum for identifying this behavior.
14. Fixed TFTP Internet Bootloader from not being accessible from Linux TFTP clients which were setting the IP header "Don't Fragment" flag bit.
15. Changed TCP so that unsent data that is automatically flushed by the `TCP_AUTO_TRANSMIT_TIMEOUT_VAL` timer includes the PSH flag. This improves GUI responsiveness for certain applications which rely on this automatic flush feature, such as the UART2TCPBridge module.
16. Fixed TCP socket loss issue which could occur if the TCP TX FIFO size was

greater than 536 bytes (TCP_MAX_SEG_SIZE). Before the fix, the socket would have gotten tied up indefinitely performing retransmissions every 1.0 seconds without detecting that the remote node was disconnected.

17.Fixed TCP socket hang issue that would occur if the PIC sent out a FIN and the remote node never responded with a corresponding FIN. The socket would have gotten stuck indefinitely in the TCP_FIN_WAIT_2 state. Thanks go to Mr. Kyle Strickland with AW North Carolina for identifying this bug.

18.Fixed UDPSetRxBuffer() function from not working if it was called before having called UDPGet() or UDPGetArray() at least once.

19.Fixed an offset error of +2 milliseconds being returned from TickConvertToMilliseconds(). Thanks go to Andrés ("saturn") on the Microchip Ethernet forum for finding this error. Note that due to integer truncation during division, this function can be off by 0.2% or so, depending on the value returned by GetPeripheralClock().

20.Updated DelayMs() macro for MPLAB C Compiler for PIC18s to work correctly when a large parameter was given. You should now be able to delay between 0 and 65535 milliseconds across all supported compilers without ending up with an unexpectedly short delay.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenabling its DHCP server.
3. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC_EXPLORER + Ethernet PICtail.
4. MAC.h RXSIZE precompiler test for proper range doesn't work. This is not a functional problem, just a compile-time configuration test. Ensure that you don't over allocate TCP_ETH_RAM_SIZE or MAX_HTTP_CONNECTIONS.

5. HI-TECH PICC-18 STD 9.51PL1 cannot compile DynDNS.c. It raises an "inconsistent type" error while trying to perform a ROM pointer to integer cast. The older 9.50PL3 compiler release is required to compile this file.

6. HI-TECH PICC-18 STD 9.50PL3 does not initialize several static variables correctly on reset. This behavior breaks many stack modules in the TCPIP Demo App and TCPIP WebVend App projects. Additionally, string printing functions do not work correctly, so the supplied "TCPIP Demo App-HITECHPICC18 PICDEMNET2 18F97J60.hex" and "TCPIP WebVend App-HITECHPICC18 PICDEMNET2 18F97J60.hex" files may not correctly print the board's DHCP assigned IP address on the board's LCD (if present) and UART. To avoid these severe problems, use the Microchip MPLAB C Compiler for PIC18 MCUs. A free student edition can be downloaded from <http://www.microchip.com/c18>.

v4.18 28 November 2007

Changes:

1. Added C32 and PIC32MX support. Some things were cleaned up in the process.
2. Removed linker scripts from C30 MPLAB projects. MPLAB IDE 8.00 can automatically select the correct linker script for 16-bit and 32-bit products.
3. Updated TCPPerformanceTest.c module. Now it automatically calculates the TX throughput and displays it for you. Also, there is now an RX throughput testing mode, which listens on a separate TCP socket (port 9763) when a TCP socket of type TCP_PURPOSE_TCP_PERFORMANCE_RX is allocated in TCPIPConfig.h. The RX socket is by default not enabled to save memory, so you must create a TCP_PURPOSE_TCP_PERFORMANCE_RX socket in TCPIPConfig.h and ensure that enough memory is allocated to accommodate it to test the RX performance test. When connected to port 9763, send a large amount of data and the PIC microcontroller will send back a count of how many bytes were received per second.
4. UDPPerformanceTest.c module now transmits 1024 packets on start up and then stops to prevent continually broadcast flooding your network. To transmit more packets after 1024 is reached, hold down BUTTON3 (left-most button on most boards).
5. Significantly improved the speed of the MD5 and SHA-1 functions. Gains for the 8-bit compilers were 50-75%, while 16-bit parts saw more modest improvements (~10%).

6. Reimplemented TCP_CLOSE_WAIT TCP state ("CLOSE_WAIT" in RFC793). Now, TCP sockets that receive a FIN from the remote node will hold off transmitting a FIN back to the remote node until the TCP_CLOSE_WAIT_TIMEOUT (defined at the top of TCP.c) elapses or immediately when the application calls the TCPDisconnect() function. This makes it possible for the application to transmit a response back to the remote node before the socket becomes closed on our end. Similarly, it simplifies application usage of the last RX bytes received as these bytes are now assured to still be in the RX FIFO for at least TCP_CLOSE_WAIT_TIMEOUT seconds. TCP_CLOSE_WAIT_TIMEOUT defaults to 200ms in this stack version.

7. Pushed the SNTP query on failure timeout up some. It was ~14 seconds and is now ~20 seconds.

8. Added TFTPOpenROMFile() API to complement TFTPOpenFile() when using PIC18 products.

9. Added a fourth parameter to newAJAXCommand() in mchp.js, allowing data to be POSTed along with the AJAX request.

10. Deprecated the TCP Loopback functions, which includes TCPOpenLoopback, TCPCloseLoopback, TCPIsLoopback, TCPInject, and TCPSteal. These functions were added in 4.10 for future SSL support, but have since become unnecessary. They are of limited usefulness, and so are being removed to save code space. The functions are still available in this version, but will be removed in the next release.

11. Added SMTPClient.ServerPort field to the SMTP API. This allows the remote server port number to be specified dynamically at run time instead of being hard coded to the SMTP_PORT value defined at the top of SMTP.c. SMTP_PORT is now only a default.

12. Added web interface to the SMTP module in the TCPIP Demo App applications.

You can now configure the SMTP module and send emails directly from within your web browser. The HTTPPostEmail() function in CustomHTTPApp.c also demonstrates how to send MIME encoded attachments in emails. The default demo will send button states, LED states, and the current potentiometer reading as a CSV file attached to the email.

13. Changed SMTPDemo() in MainDemo.c to trigger on BUTTON2 and BUTTON3 simultaneously held down instead of BUTTON0 only.

Fixes:

1. Fixed an ENC28J60.c MACGetArray() bug which would overwrite one byte of memory at address 0xFFFFFFFF if you provided NULL for the destination address pointer.

2. Fixed an MPFS2.c MPFSGet() bug which would overwrite memory address 0x00000000 if a NULL pointer was provided as the destination.
3. Fixed a bug in the HTTP2 server accessing incorrect sockets if an inadequate number of sockets were available on POR.
4. Fixed Internet Bootloader project from failing with a timeout if an ARP packet arrived during the Erase/Write operation.
5. Fixed DHCP client RFC non-compliance where it would send the ciaddr field in the initial SELECTING state. Also, in the RENEWING state, the Requested IP Address option was being sent, which is illegal. These changes may fix compatibility problems with certain DHCP servers.
6. Fixed TFTP Client's TFTPcloseFile() function from sending data using a wrong UDP socket if StackTsk() was called after TFTPisFileOpened() was last called.
7. Added two zero bytes to the ICMP echo request payload to improve compatibility with some buggy NAT routers.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenabling its DHCP server.
3. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC_EXPLORER + Ethernet PICtail.
4. MAC.h RXSIZE precompiler test for proper range doesn't work. This is not a functional problem, just a compile-time configuration test. Ensure that you don't over allocate TCP_ETH_RAM_SIZE or MAX_HTTP_CONNECTIONS.

v4.16 06 November 2007

Changes:

1. Added Internet Radio application. This is a TCP client application which downloads streaming MP3 audio from a Shoutcast server and then plays it back to stereo earphones via a VLSI VS1011 audio decoder.
2. Added SPIRAM.c module. This module is intended for interfacing to an AML Semiconductor N256S0830HDA SPI RAM chip. The TCP module can now interface directly to this SPIRAM module to store TCP socket FIFO buffers and other TCB data in the external RAM.
3. Added TCP_OPTIMIZE_FOR_SIZE compile time configuration macro to TCP.c file. When optimizing for small code size, the TCP module ROM footprint shrinks up to 6KB, but performance may slow down on some processors (namely PIC18s, where the penalty is approximately 15%).
4. Added USE_EEPROM_25LC1024 compile time configuration macro to TCPIPConfig.h. Enable this definition if you are storing your MPFS[2] on a 1Mbit 25LC1024 or similar EEPROM device that uses 24-bit addressing and a 256 byte write page size.
5. Changed LCDBlocking.c module initialization code. It should now be possible to use 4-bit mode on certain "unusual" LCD controllers, like the Samsung S6A0032. Most PICDEM.net 2 and Explorer 16 boards use an LCD with this controller.
6. SNTP client now attempts to requery the SNTP server about every 14 seconds if the last query attempt fails. This allows the internal time value to become valid quickly should the board be powered up before an Ethernet cable is attached or if the DHCP client doesn't obtain an IP address quickly enough. Previously, it would take up to 10 minutes after plugging the Ethernet cable in to get a correct time value from the SNTP server.
7. Added UDP_USE_TX_CHECKSUM compile time configuration macro to TCPIPConfig.h. When enabled, all UDP packets will have a correct UDP checksum computed and inserted into the UDP header of outbound packets. If you do not define this macro, the UDP checksum will be disabled (left as 0x0000), which is how previous stack versions operated. Note that enabling checksum generation cuts your maximum UDP TX throughput by nearly half due to the required computations.
8. Substantially changed TCP socket RX and TX FIFO allocation. Now, sockets can be stored either in Ethernet RAM, PIC RAM, or external (SPI) RAM. Previously, sockets could only be allocated in Ethernet RAM, which was not scalable.
9. Added TCPOpen() API function. This replaces TCPListen() and TCPConnect(). TCPOpen() supports a large number of options that will make the creation of client mode sockets much easier. You can specify the remote node as a hostname that needs DNS and ARP resolution, an IP address that only needs ARP resolution, or legacy NODE_INFO pointer for direct compatibility with the previous

TCPListen() and TCPConnect() APIs. TCPOpen() also supports a socket type parameter which will allow you to use the new TCP socket RAM allocation system.

10. Added TCP Keep Alive mechanism defined by RFC 1122 section 4.2.3.6 to the TCP module. This helps automatically detect lost connections. If the remote node sends back an RST, this immediately closes the lost connection on our end. Currently, no action is taken if the keep alive gets no response. Note that this feature deviates from the standard by defaulting to only 10 seconds instead of over 2 hours. Also deviating from the standard, this feature is enabled by default. To disable it, undefine TCP_KEEP_ALIVE_TIMEOUT at the top of TCP.c.

11. Moved TCPPerformanceTest.c module from default port 12345 to 9762.

12. Moved UDPPPerformanceTest.c module from default port 12345 to 9, the "discard" protocol port.

Fixes:

1. The DHCP client now specifically requests the previous IP address when a DHCP renewal occurs.
2. The SNTP client now correctly maintains time when repetitively calling SNTPGetUTCSeconds() between an NTP requery event. Thanks go to Rob Haverkort on the Microchip Ethernet forum for noticing the time value incrementing far faster than it should have.
3. TCP module will not transmit a bunch of unnecessary duplicate ACK packets when data is ready to be transmitted but the remote RX window is zero. This previously didn't cause anything to break, but would waste CPU time and bandwidth sometimes.
4. TCP sockets will no longer automatically close if the remote RX window stays zero for several seconds.
5. Fixed TFTP Internet Bootloader project from corrupting the configuration fuses. Previously, this would result in the Watchdog timer being enabled and causing an unintentional reboot every few minutes with the demo TCP/IP stack.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. TFTPc module has not been tested with this version.
3. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable

their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.

4. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC_EXPLORER + Ethernet PICtail.

5. MAC.h RXSIZE precompiler test for proper range doesn't work. This is not a functional problem, just a compile-time configuration test. Ensure that you don't over allocate MAX_TCP_SOCKETS, TCP_TX_FIFO_SIZE, TCP_RX_FIFO_SIZE, or MAX_HTTP_CONNECTIONS.

v4.13 02 October 2007

Changes:

1. Added command line support to the MPFS2.exe tool. You can now generate MPFS output files using batch scripts or other console applications.
2. Added dynamic variable parameter capabilities to the MPFS2 utility. To use, add the parameters you wish to pass to the end of the dynamic variable. All parameters are passed as WORD values. (ex: ~myArray(2,5)~)
3. Added TCPWasReset() API to allow the application layer to be notified if an underlying socket reset has occurred (ex: remote node disconnects, cable is disconnected and times out, user calls TCPDisconnect(), etc.). The reset state is latching, which allows the application layer to detect if a remote node disconnects and a new connection occurs on the same socket before the application can detect the original disconnection through the TCPIsConnected() API.
4. Added a counter to the UDPPerformanceTest module and made it suppress transmission if an Ethernet link is not present.
5. Added TCPIP WebVend App example application to the main stack distribution. This corresponds to three new Microchip Webinars being published on the HTTP2 server usage topic.

Fixes:

1. Fixed MPFS2.exe PC utility from crashing if you attempt to generate an MPFS classic .bin/.c/.s output file.
2. Fixed RCONbits definition for HPC_EXPLORER hardware profile when using the HI TECH

PICC-18 compiler.

3. Fixed a MPFSGetFilename() bug when using C30 and MPFS2 images stored in program memory. Thanks to Billy Walton on the Microchip Ethernet forum for identifying this issue.

4. Fixed a TCP RX FIFO corruption problem which would occur if the remote node sent more data than could fit in our RX FIFO in a single packet. The GeneticTCPClient.c module was subject to experiencing this problem when connected to www.google.com's servers.

5. Fixed a DHCP client UDP socket leak if you called DHCPDisable() after the DHCP client had already obtained a UDP socket. Thanks go to Matthew Kendall on the Microchip Ethernet forum for identifying this problem.

6. Fixed a SNMP Server module bug testing a string length (with respect to SNMP_COMMUNITY_MAX_LEN) being off by one, resulting in possible memory corruption. Thanks go to Matthew Kendall on the Microchip Ethernet forum for identifying this problem.

7. Cleaned up some C30 compiler warnings related to macro definitions with inadequate parenthesis in them.

8. Fixed HTTP2 module sometimes returning a 501 error instead of a correct web page when being bombarded with new connection requests.

9. Fixed a TickGet*() API problem where the returned tick value could be off by 64K ticks occasionally on PIC24 and dsPIC processors.

10. Fixed SMTP client module failing to send email when attempting to send an email with a 'CC' or 'BCC' field that was in ROM while the 'To' field was in RAM or visa versa.

11. Fixed TCP module sending an incorrect sequence number in RST packets sent when in the TCP_SYN_SENT state and an invalid segment arrives. In prior stack versions, some TCP client applications might take a very long time to recover in the event of a power failure, reset, and subsequent reconnect to a remote server that still thinks the old connection is still active. With this fix, reconnections should be possible almost immediately after a power failure because the correct RST packet will cause the old connection to get closed right away.

12. Fixed a TCP socket leak problem that would occur over if the local PIC called TCPDisconnect() and the remote node didn't send us a correct FIN response. Sockets could previously get lost in the TCP_FIN_WAIT_2 state and wouldn't recover unless the application called TCPDisconnect() a second time with the same socket handle.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. TFTPc module has not been tested with this version.
3. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.
4. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC_EXPLORER + Ethernet PICtail.
5. HI-TECH PICC-18 projects will not correctly set the processor configuration fuses through code using the `__CONFIG()` macro. Ensure that the configuration fuses are manually set correctly via the MPLAB IDE Configuration Bits dialog. This problem has been observed with compiler version 9.50PL3.
6. MAC.h RXSIZE precompiler test for proper range doesn't work. This is not a functional problem, just a compile-time configuration test. Ensure that you don't over allocate `MAX_TCP_SOCKETS`, `TCP_TX_FIFO_SIZE`, `TCP_RX_FIFO_SIZE`, or `MAX_HTTP_CONNECTIONS`.
7. GenericTCPClient example of downloading a web page from www.google.com is extremely slow. The default TCP socket has too little RX space to accept a full packet sent from Google's servers, so the remote server must retransmit a lot of data, slowing the transfer down a lot. Making `TCP_RX_FIFO_SIZE` 536 bytes or bigger and correspondingly shrinking `MAX_TCP_SOCKETS` will correct this problem.

v4.11 27 August 2007

IMPORTANT NOTE: You must use MPLAB 7.62 or higher to successfully open the MPLAB projects.

Changes:

1. Added a Microchip TCP/IP Stack Users' Guide to document the stack

features/modules/and APIs and address the stale AN833 documentation. Note that this is a work in progress. Many modules have yet to be documented in the Users' Guide.

2. Added HTTP2 module. This HTTP module includes a whole new API and supreme new features, such as POST support, cookies support, browser authentication support, and more.

3. Added MPFS2 module. This module is required for the new HTTP2 module and performs better while having fewer limitations. Long filenames and folders are now supported.

4. Added a new GUI based MPFS2.exe PC utility. The older MPFSv2.exe GUI application and MPFS.exe command line tool has been retired. The new utility has advanced features, such as MPFS2 file format support, GZIP compress, etc.

5. Added a TFTP bootloader. This is a stand alone project and currently only supports the PIC18F97J60 family of PIC processors with internal Ethernet.

6. Added UART2TCPBridge.c file and `STACK_USE_UART2TCP_BRIDGE` option to `TCPIPConfig.h`. This new module acts as a TCP and UART bridge, with a high priority UART interrupt and dedicated UART TX and RX FIFOs for minimum UART latency and maximum performance. By default, the bridge acts as a TCP server and listens on port 9761. The UART baud rate defaults to 19200. The bridge can be reconfigured to act as a TCP client.

7. Added Simple Network Time Protocol (SNTP) client. This module automatically obtains the current time (date) from the Internet. Enable this module by defining `STACK_USE_SNTP_CLIENT` in `TCPIPConfig.h`. Obtain the current time (in seconds since 00:00:00 1970) by calling the `SNTPGetUTCSeconds()` API.

8. Added support functions `Base64Encode()` and `Base64Decode()` in `Helpers.c`. Base 64 is required for the new HTTP2 module, but of general use to many applications.

9. Added SMTP Authentication support to the SMTP Client. To use this, set the `SMTPClient.Username` and `SMTPClient.Password` string pointers to a non-NULL value before calling `SMTPSendMail()`. Applications implementing email transmission capabilities should expose these options to the end-user for configuration. To use SMTP servers that do not support the AUTH LOGIN authentication command, simply leave the `SMTPClient.Username` and `SMTPClient.Password` pointers as their default NULL value.

10. Converted `DHCPDisable()` from a macro to a real function and added the complementary `DHCPEnable()` function. These two functions can be used at run time to dynamically switch between using a static IP address and configuration and DHCP assigned IP address and configuration.

11. Updated `StringToIPAddress()` to work more robustly, including the ability to decode host name strings and determine if they contain a valid IP address or not. Also, the complementary `ROMStringToIPAddress()` function was added.

12. Updated the DNS module. Now, if you give it an IP address string to resolve, it will convert the string to an IP address and immediately return without querying the DNS.
13. Shrunk the advertised TCP Maximum Segment Size from 576 bytes to 528 bytes. This might improve compatibility if your TCP data has to propagate over nodes with small MTUs and you have a correspondingly large TCP RX FIFO defined.
14. Performed some maintenance on the FTP.c file. No significant functionality has been changed, but some potential problems were corrected.
15. Altered Tick.c file and API. Now, the Tick module can operate maximum precision, returning the value of the actual Timer as it is counting, without disturbing the timer count by writing to it or disabling it. Three new APIs were added, TickGetDiv256(), TickGetDiv64K(), and TickConvertToMilliseconds(). Internally the tick counter is now 48-bits wide and as accurate as your Timer clock source, allowing you to use it as a Real Time Clock.
16. Added PIC24FJ64GA004_PIM hardware profile. This hardware profile is intended for use with the PIC24FJ64GA004 PIM on the Explorer 16 development board. In this mode, BUTTON2 and BUTTON3 and several of the LEDs do not work correctly due to lack of I/O pins on this device. Also, you cannot have the POT and TEMP jumpers on the PIM bridged because these signals are multiplexed with the SDO1/SDI1 pins needed for the Ethernet PICtail Plus.
17. Removed most ROM APIs when using a 16-bit compiler (C30). PIC24s and dsPICs usually don't need separate ROM functions since the Program Space Visibility feature maps ROM into RAM space. All ROM APIs are still supported, but they are now macros to base RAM APIs. This change saves a couple of kilobytes of code space on PIC24 and dsPICs.
18. Improved MyTCB structure caching. This should reduce TCP packet processing overhead with the ENC28J60 where TCBs are stored in the Ethernet RAM.
19. MAX_RETRY_COUNTS TCP configuration option has been renamed to TCP_MAX_RETRIES.
20. FTP server is no longer enabled by default. HTTP2 now supports POST, so you can upload new webpages through the /mpfsupload page now. FTP required two precious TCP sockets.
21. Began adding hooks for an SSL/TLS transport for secure HTTPS and other future stack modules. Note that these cryptographic modules are not available at this time. Configuration options such as MAX_SSL_CONNECTIONS do nothing and should not be modified.
22. Username has changed for all of the modules. Now all modules have a default username of "admin" and password of "microchip". Previously, the FTP and Telnet modules used "ftp" and "telnet" respectively for the usernames.

Fixes:

1. Fixed a SendFile() bug in HTTP.c where parsing dynamic cgi files could send garbage back to the web browser sometimes. Thanks go to Matt Watkins on the Microchip Ethernet forum for identifying this issue.
2. Fixed an off by one error in the calculation of RESERVED_TCP_MEMORY. Previously, the last TCP socket's RX FIFO would incorrectly overlap with the Ethernet RX buffer, causing incoming packets to occasionally be corrupted or the incoming data on the last socket to get corrupted.
3. Fixed the QWORD_VAL's dword struct element types. dword.LD and dword.HD were incorrectly defined as WORDs instead of DWORDs. Thanks go to Iñaki Esparza on the Microchip Ethernet forum for identifying this issue.
4. Fixed the incorrect processing of received IP fragments with a non-zero offset. This stack does not support IP packet reconstruction due to the limited amount of available RAM. Thanks go to Iñaki Esparza on the Microchip Ethernet forum for noticing this behavior.
5. Board now only responds to ping requests to our IP address, the directed subnet broadcast address, or the broadcast address of 255.255.255.255. Previously, it would respond to any ping request to any IP address, assuming the MAC address was correct.
6. Fixed a memory corruption/UDP packet loss problem when handling incoming UDP packets. Previously, StackTask() would incorrectly continue processing more packets if it came upon a UDP packet. Thanks go to Iñaki Esparza on the Microchip Ethernet forum for identifying this issue.
7. Fixed the SMTPClient.ROMPointers.Server flag having an inverted meaning. Previously, the SMTP client module would treat the SMTPClient.Server pointer as a ROM pointer if this bit was cleared. In most cases, this would cause the SMTP client to return an error code of 0x8000 when the SMTPClient.SMTPServer address pointer was set.
8. Fixed the DHCP Server module from incorrectly parsing received packets which had a DHCP_PARAM_REQUEST_IP_ADDRESS option followed by more options. Previously due to the length miscalculation, the parser would enter a random state, depending on the packet's contents. Thanks go to Iñaki Esparza on the Microchip Ethernet forum for identifying this issue.
9. Fixed potential incorrect results when UDPIsGetReady() was called and a previous application did not call UDPDiscard() on an RX packet. Now, StackTsk() calls UDPDiscard() as appropriate to let it know when it's old RX data is being thrown away. This fixes a potential bug in the DHCP Server module and makes the UDP API

more robust. Thanks go to Iñaki Esparza on the Microchip Ethernet forum for identifying the potential DHCP server issue.

10.Fixed a potential ARP bug where the Gateway's MAC address would be returned for an IP address on the local subnet. This unusual case would occur when two application tasks were using the ARP module at the same time and the second application was trying to resolve an IP address off of our subnet. Thanks go to Iñaki Esparza on the Microchip Ethernet forum for pointing this issue out.

11.Fixed an PIC18F97J60 family MAC layer bug where MACGetArray() might not correctly increment the Ethernet read pointer if a NULL pointer was given for the destination. The C compiler might have optimized the function so that it would increment the read pointer one less than it was supposed to.

12.The TCP module now acknowledges TCP Keep-Alive packets which will help prevent connection loss if the remote node fills up our RX FIFO and then our window-update packet gets lost on the network/Internet. In stack version 4.02, a zero-window probe would have been required to restore the communications.

13.Fixed a TCP RX FIFO corruption issue that would occur in (uncommon) circumstances when too many out-of-order segments arrived such that a second "hole" would have been required to accommodate the data. Thanks go to Iñaki Esparza and his eagle eyes on the Microchip Ethernet forum for finding this corner case bug.

14.Inline assembly in the ETH97J60.c file has been modified to accommodate the C18 Extended mode and C18 Auto default storage class. Previously, the Ethernet module would transmit garbage packets when using the C18 parameter stack.

15.Fixed potential buffer overflow in NBNS.c's NBNSGetName() function where an unexpected string length retrieved from the packet could cause random memory corruption.

16.Fixed some potential PIC18F97J60 family Ethernet module transmit lockup conditions that occur on some networks. Previously blocking while() loops would wait indefinitely for the ECON1<TXRTS> bit to become clear by hardware, which the hardware might never have done.

17.In MainDemo.c, a call to DelayMs() was being made using a value of 100ms. This was too long for the underlying Delay1KTCYx() C18 function and would result in a shorter than expected delay when compiled with C18. This has been fixed with a loop. Thanks go to Andy123 on the Microchip Ethernet forum for pointing this problem out.

18.Fixed a potential C18 memory overlaying problem in the TickUpdate() function. Previously, the local variable used in this function might have been overlayed on other memory, resulting in random memory corruption as the ISR occurred.

19.The demo AJAX web pages in the TCPIP Demo App\WebPages folder now correctly display

and self-refresh in Firefox 2. Previously, it would work in Firefox 1.5 and

Microsoft Internet Explorer, but not Firefox 2. Thanks go to "gohsthb" on the Microchip Ethernet forum for identifying this correction.

20. Rewrote the GenericTCPServer.c example to not use an application RAM FIFO for buffering. Since the TCP module implements its own FIFOing, the application has limited need for its own FIFO too. This fixes a previous bug where the GenericTCPServer was not checking the number of incoming bytes with the remaining size available of the App FIFO. This would have previously resulted in a buffer overflow, corrupting the RX data if too much arrived all at once.

21. Fixed a potential MPFS classic inline ASM30 assembly code problem where web pages stored in internal Flash and C30 with optimizations enabled could result in data corruption.

22. Fixed a UDPPut() tracking problem that would result in extra bytes being appended to the end of a packet if the UDPSetTxBuffer() function was used. This previously caused the SNMP module to send some junk data at the end of its packets.

23. Fixed a potential TCP problem where transmitted FIN packets might not get retransmitted properly if the remote node never acknowledged the data that was transmitted just before the FIN was sent.

24. Fixed a NetBIOS Name Service bug where the response packet would sometimes get sent to an incorrect address. It now consistently responds to the unicast MAC/IP address of the NBNS query packet.

25. Added padding to all transmitted DHCP messages to make the minimum UDP payload at least 300 bytes. This fixes compatibility with some older BOOTP relay devices which discard smaller packets. Thanks go to Dave Collier on the Microchip Ethernet forum for pointing this problem out.

26. Substantially shrunk the number of retransmission attempts made in the TCP_SYN_RECEIVED state. This improves recovery time when attacked by a SYN flood Denial of Service event. The recovery time is now 7 seconds (3 total packets) instead of 31 seconds (6 total packets)

27. Fixed the possibility of the NetBIOS Name Service module giving out the board's static IP address before a DHCP lease could be obtained. NBNS requests are now only serviced when originating from nodes on the same subnet.

28. Fixed storage of MPFS classic in internal program memory when using the HI-TECH PICC-18 compiler.

29. Substantially revised TCP.c, fixing many TCP bugs and possibly adding new ones. Thanks go to Michael Rubinstein for finding several of these TCP problems.

30. The DNS client module will now time out and return failure if the DNS server cannot be ARPed or does not respond to the DNS query. Each timeout is set to 1 second and

3 total ARP and 3 total DNS query attempts are possible. Previously, it would retry indefinitely, causing the calling application to deadlock.

Known Problems:

1. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
2. TFTPc module has not been tested with this version.
3. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to reenable it's DHCP server.
4. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC_EXPLORER + Ethernet PICtail.
5. HI-TECH PICC-18 projects will not correctly set the processor configuration fuses through code using the `__CONFIG()` macro. Ensure that the configuration fuses are manually set correctly via the MPLAB IDE Configuration Bits dialog. This problem has been observed with compiler version 9.50PL3.

Testing and Performance Notes:

1. Make sure to use MPLAB IDE 7.62 or higher with this version. Versions below 7.61 will not work. Version 7.62 has cool new features like C auto-word complete and function parameter tooltips that can be enabled (disabled by default).
2. Testing was done using MPLAB C18 version 3.12, MPLAB C30 version 3.01, and HI-TECH PICC-18 version 9.50PL3. Make sure to upgrade your tools to at least these versions.

v4.02 10 April 2007

IMPORTANT NOTE: You must use MPLAB 7.41 or higher to successfully open the MPLAB projects.

IMPORTANT NOTE2:If an external serial EEPROM memory is used to store AppConfig, it's contents will be invalidated the first time you run this version, restoring the AppConfig defaults. The AppConfig structure has been optimized.

IMPORTANT NOTE3:If an external serial EEPROM memory for MPFS, you will need to recreate the MPFS image and program your EEPROM. A 32 bit addressing format is now used.

Changes:

1. Implemented TCP RX packet order correction logic. The stack can now accept TCP frames that arrive out-of-order without requiring the remote node to go through a retransmit cycle. This dramatically improves RX performance when communicating over the Internet.
2. UDPOpen() now can handle a NULL pointer for remoteNode. In this case, the broadcast IP/MAC addresses will be used for the remoteNode (destination address of outbound packets).
3. Recreated MPLAB projects for the HI-TECH PICC-18 compiler. These were temporarily absent from 4.00RC. This project works with the PIC18F97J60 with internal Ethernet module, assuming the correct compiler version is present.
4. Moved all the headers around. Most of them are in "Microchip Solutions\Microchip\Include\TCPIP Stack" now. This change was made to again be more compatible with other (future) Microchip software libraries.
5. New UDPPut() behavior. Now, if space in the Ethernet TX buffer runs out, the packet will not automatically be transmitted. You must call UDPFlush() to cause the packet to be transmitted.
6. Added UDPGetArray(), UDPPutArray(), UDPPutROMArray(), UDPPutString() and UDPPutROMString() user API functions. These functions perform substantially better than calling UDPPut() successively and allow greater application programming flexibility.
7. Changed TCPPutString() and TCPPutROMString() APIs to now return an updated string pointer instead of a count of bytes successfully placed in the TX buffer.
8. Added UDPPerformanceTest.c. By default this module causes UDP packets containing 1024 bytes of application data to be broadcasted on UDP port 12345. Use a packet sniffer, such as Wireshark (<http://www.wireshark.com/>) to capture and derive stack overhead/UDP TX performance characteristics with

this module. Note that this test uses the UDPPutROMArray() function.

Applications which use successive calls to UDPPut() will be slower. To

enable this module, #define STACK_USE_UDP_PERFORMANCE_TEST in TCPIPConfig.h.

9. Added TCPPerformanceTest.c. By default this module listens on TCP port

12345. When a remote client connects, this server module will begin

transmitting the maximum possible amount of application data that it can,

given your TCP TX FIFO size. Use a packet sniffer, such as Wireshark

(<http://www.wireshark.com/>) to capture and derive stack overhead/TCP TX

performance characteristics with this module. Any TCP client can be used,

including readily available utilities such as the telnet.exe utility

available on Microsoft Windows XP. To use it to connect to the test module,

run: "telnet.exe xxx.xxx.xxx.xxx 12345" where xxx.xxx.xxx.xxx is the board's

IP address. Note that this test uses the TCPPutROMArray() function.

Applications which use successive calls to TCPPut() will be slower. To

enable this module, #define STACK_USE_TCP_PERFORMANCE_TEST in TCPIPConfig.h.

10. Added Reboot.c module. By default, this module listens on UDP port 30304.

If the application byte 0x00 arrives on this port, the PIC will reset. This

is primarily useful for remote Bootloader entry.

#define STACK_USE_REBOOT_SERVER in TCPIPConfig.h to enable this module.

Note that since no encrypted challenge/response algorithm is currently

implemented, this module is a Denial of Service vulnerability, so it should

not be enabled unless there is a specific need for it.

11. Made the TickUpdate() ISR routine execute in the low priority ISR instead of

the default high priority ISR. The Microchip TCP/IP stack does not need any

interrupts except this low priority timer.

12. Renamed STACK_USE_DHCP macro to STACK_USE_DHCP_CLIENT

13. Added STACK_USE_MPFS macro.

14. Changed UDPISPutReady() to return a WORD instead of a BOOL. The WORD is the

number of bytes that can be put into the buffer.

15. Changed MACGetArray() to accept a NULL pointer. If NULL, the retrieved data

will simply be discarded. This also changes the behavior of UDPGetArray()

and TCPGetArray() to match, throwing bytes away if a NULL pointer is given.

16. Added a very simple DHCP Server module. This module has limitations and is

useful for a single client only. Its purpose is to allow you to directly

connect the board to a standard PC through a crossover cable (no other

network nodes attached). The server is coded to automatically disable

itself if the DHCP client is also enabled and another DHCP server is

detected on the network. This allows both the DHCP server and DHCP client

to coexist without any manual reconfiguration.

17.Added DNSResolveROM() function for resolving host names that are stored in program memory, ex: literal strings.

18.Added a TCP automatic transmit/window update timer. It defaults to TCP_AUTO_TRANSMIT_TIMEOUT_VAL (40ms) after the first get or put operation following the last automatic transmit/window update. This timer enhances performance, especially when streaming data over the Internet where round trip times can be several tens to low hundreds of milliseconds. This also improves application coding flexibility as TCPFlush() need not be called anymore.

19.Added TCP delayed ACKnowledgement timer. This conserves bandwidth by transmitting fewer ACKs and prevents inadvertently influencing remote slow start/collision avoidance and fast retransmit algorithms.

20.Completely rewrote ICMP (ping) server module. It is now much smaller (ROM and RAM), faster, and can handle packets of 576 bytes or larger, if no IP fragmentation occurs.

21.Rewrote StackTsk() stack manager. It is much simpler now.

22.Added TCPFind(), TCPFindArray(), and TCPFindROMArray() user API functions.

These functions peek inside a given TCP socket's RX FIFO (without removing anything) and looks for a particular byte or array of bytes. This should greatly simplify the creation of application code whenever variable length fields are used (ex: text strings terminated by \r\n). It supports case insensitive text searching or binary searching, as well as an offset to start searching at.

23.Added TCPGetRxFIFOFree() user API. It returns the number of bytes of free space in the TCP's RX FIFO.

24.Changed default TICK resolution to 1ms (from 10ms) and improved accuracy.

25.Added outbound ping capabilities (i.e. board can now ping another board or a PC). To enable these features, define STACK_USE_ICMP_CLIENT. This will enable several new APIs, including ICMPBeginUsage(), ICMPSendPing(), ICMPGetReply(), and ICMPEndUsage(). The functions should be called in this order. See the PingDemo() function in MainDemo.c for an example of how to use them. By default, pushing BUTTON3 (left-most one) will cause a ping to be sent to 4.78.194.159 (ww1.microchip.com). The response time will be displayed on the LCD (assuming your development board has an LCD).

26.Cleaned up C30 3.00 signed/unsigned warnings.

27.Removed PIC18F97J60_TEST_BOARD hardware profile support. This stack no longer supports it due to the old beta silicon (with errata) mounted on

these boards.

28. Added support for ROM pointers for all of the SMTP strings (To, From, CC, Subject, etc.). If you use a ROM string, you must also set the corresponding SMTPClient.ROMPointers.xxx bit to let the SMTP module know which type of pointer was provided. See the SMTPDemo() code in MainDemo.c for an example calling sequence using both ROM and RAM strings for the various fields.

Fixes:

1. Fixed a critical TCP buffer corruption issue where the start of a TCB header overlapped with the last byte of the RX FIFO from the previous socket. This bug affected version 4.00RC only.
2. ETH97J60.c, TCPIP.h, and TCPIP Stack Version.txt were correctly read to the TCPIP Demo App-C18 project using relative paths instead of absolute paths.
3. UDPOpen() now dynamically assigns a local port number if you call it and give it a 0x0000 port number. This should fix some UDP applications from not working (ex: DNS Client module) with some computers/routers/networks which throw away traffic originating from the invalid port 0x0000 value.
4. Fixed a ENC28J60 bank selection error that would occur if an application called GetCLKOUT() in ENC28J60. By default, this function is not called.
5. UnencodeURL() function in Helpers.c is now tested and working.
6. Fixed a TCP Window Update problem when TCPGetArray() was used. Before the problem was fixed, performance could have been terrible on reception.
7. Fixed a unintended TCP connection close if the socket was idle for about a minute. Now, TCP sockets will remain open indefinitely if there is no traffic going on.
8. Serial numbers >32K are now displayed correctly on the serial port as a positive value when C18 is used and the board is placed in configuration mode (BUTTON0 is depressed on power up).
9. HI-TECH PICC-18 compiler would previously incorrectly initialize the AppConfig structure.
10. Previously a processor reset was possible when accessing items in the AppConfig structure on 16 bit MCUs (PIC24, dsPIC) due to unaligned word accesses. This was fixed by reordering the Flags byte in the APP_CONFIG structure.
11. Rewrote DHCP client state machine, fixing the previously known problem where it would not perform a new discovery if it was trying to renew a lease

with an offline DHCP server.

12.Fixed a critical deadlock problem in the ETH97J60.c MAC layer driver for the PIC18F97J60 family Ethernet controller. Previously, it was possible (although rare) that the DMAST or TXRTS bits would get stuck set if too much Ethernet traffic was received within a short interval. Previously, the MACFlush() function was unnecessarily setting TXRST, which it should not do while the Ethernet interface or DMA is being used.

13.Fixed an HTTP server state machine problem where a new connection occurring too soon on a previously used socket could cause the HTTP server to no longer respond.

14.Fixed a potential memory corruption error in the HTTPGetVar() callback which would exceed the bounds of the VarString array when returning the VAR_STACK_DATE variable.

15.Fixed a TCP transmission sequence tracking problem whenever data is retransmitted and new unflushed data is also in the TX FIFO. Thanks go to Matt Watkins on the Microchip Ethernet forum for identifying this issue.

Known Problems:

1. RTL8019AS MAC layer driver has not been updated for new TCP module. Users requiring RTL8019AS support should continue to use stack version 3.75.
2. I2CEEPROM.c has not been tested or completed. Continue to use I2CEEPROM.c from stack version 3.75 if this file is needed.
3. Telnet server module does not implement a lot of Telnet functions. As a result, it will likely not display correctly or work at all with some Telnet clients. The server was tested with the Microsoft telnet.exe utility which is provided with Microsoft Windows.
4. TFTPc module has not been tested with this version.
5. The default demo web pages which use AJAX do not automatically refresh themselves when viewed in Firefox 2.0.0.1. Earlier Firefox versions (1.5ish) probably work without any problem.
6. Files may be inaccessible in your MPFS if compiled with C18 for internal flash program memory and your total MPFS content is large (around 64KB or larger). The code attempts to access the ROM memory using a near rom pointer when a far rom pointer is needed.
7. If using MPLAB 7.52 all .s files that are compiled with C30 will not have the corresponding object file get stored in the correct directory. As a result, if you are compiling with C30 and with MPFS_USE_EEPROM not defined (i.e. storing web pages in internal program memory), the project won't link

(throws a undefined reference to `MPFS_Start'). As a workaround, remove the Intermediates Directory in the MPLAB project. Alternatively upgrade MPLAB to a newer version. MPLAB IDE 7.60+ may have this fixed.

8. If the DHCP client and DHCP server are used at the same time and you connect two similar boards to each other (ex: two PICDEM.net 2 boards connected via a crossover cable), a race condition can occur where both nodes will disable their DHCP server and neither board will get a successful DHCP lease. If this unlikely scenario occurs, as a work around, simply reset one of the boards to enable it's DHCP server.

9. HI-TECH PICC-18 projects may not compile when MPFS_USE_EEPROM is not defined and you are trying to store web page data in internal FLASH program memory.

10. HI-TECH PICC-18 projects may not compile when targeting the external ENC28J60 chip on the PICDEM.net 2 development board (instead of the internal Ethernet controller). This problem only applies when a PIC18F97J60 family part is the target. I.e. it compiles correctly for the HPC_EXPLORER + Ethernet PICtail.

Testing and Performance Notes:

1. This stack version was compiled and tested with the following tool versions:

-MPLAB IDE 7.52

-Microchip C30 version 3.00

-Microchip C18 version 3.10

-HI-TECH PICC-18 version 9.50PL3

2. Using the UDPPerformanceTest.c module, the stack can transmit around 220KBytes/second (1.75Mbits/second) of UDP application data on the PIC18F97J60 with internal Ethernet @ 41.66667MHz core clock, compiled using C18 3.10 with debug optimization settings.

3. Using the UDPPerformanceTest.c module, the stack can transmit around 392KBytes/second (3.14Mbits/second) of UDP application data on the PIC24HJ256GP610 with external ENC28J60 @ 40 MIPS, compiled using C30 3.00 with debug optimization settings.

4. Using the TCPPerformanceTest.c module, the stack can transmit around 58KBytes/second (464Kbits/second) of TCP application data on the PIC18F97J60 with internal Ethernet @ 41.66667MHz core clock, compiled using C18 3.10 with debug optimization settings, over Ethernet when using a tiny 200 byte TX TCP FIFO. Note that performance can be improved significantly by increasing the FIFO size and performance will drop significantly if the

round trip TCP acknowledgement time is increased (ex: testing over the Internet instead of Ethernet).

5. Using the TCPPerformanceTest.c module, the stack can transmit around 69KBytes/second (558Kbits/second) of TCP application data on the PIC24HJ256GP610 with external ENC28J60 @ 40 MIPS, compiled using C30 3.00 with debug optimization settings, over Ethernet when using a tiny 200 byte TX TCP FIFO. Note that performance can be improved significantly by increasing the FIFO size and performance will drop significantly if the round trip TCP acknowledgement time is increased (ex: testing over the Internet instead of Ethernet).

6. Using the TCPPerformanceTest.c module, the stack can transmit around 178KBytes/second (1.42Mbits/second) of TCP application data on the PIC24HJ256GP610 with external ENC28J60 @ 40 MIPS, compiled using C30 3.00 with debug optimization settings, over Ethernet when using a larger 2000 byte TX TCP FIFO. Note that performance will drop significantly if the round trip TCP acknowledgement time is increased (ex: testing over the Internet instead of Ethernet).

v4.00RC 28 December 2006

IMPORTANT NOTE: If an external serial EEPROM memory is used to store AppConfig, it's contents will be invalidated the first time you run this version, restoring the AppConfig defaults. The AppConfig structure has been optimized.

IMPORTANT NOTE2: If an external serial EEPROM memory for MPFS, you will need to recreate the MPFS image and program your EEPROM. A 32 bit addressing format is now used.

Changes:

1. Added Simple Mail Transfer Protocol (SMTP) client module and updated MainDemo.c to exercise the Email transmission functionality when a user pushes BUTTON0.
2. Added beta Telnet server module. See Known Problems section.
3. Completely revamped the TCP module. A real transmit FIFO and receive FIFO are allocated for each TCP socket now. This greatly enhances RFC compliance, communications robustness, and makes application development easier. New APIs were added for putting and getting arrays and strings (including ROM

variants). Several TCP related bugs are now fixed as a result. Please report any bugs found in the new implementation.

4. Added TCPPutArray() API.

5. Added TCPPutROMArray() API.

6. Added TCPPutString() API.

7. Added TCPPutROMString() API.

8. Added TCPGetArray() API.

9. Changed TCPIsPutReady() API. Instead of returning a BOOL, it now returns a WORD. The WORD is a count of the number of bytes that TCPPut(), TCPPutArray(), etc. can immediately place in the output buffer. MAKE SURE THAT YOUR CODE DOES NOT COMPARE THE RETURN RESULT OF TCPIsPutReady() DIRECTLY TO TRUE. For example, "if(TCPIsPutReady(MySocket) == TRUE){...}" must be converted over to: "if(TCPIsPutReady(MySocket)){...}".

10.Changed TCPIsGetReady() API. Instead of returning a BOOL, it now returns a WORD. The WORD is a count of the number of bytes that TCPGet() or TCPGetArray() can immediately obtain. MAKE SURE THAT YOUR CODE DOES NOT COMPARE THE RETURN RESULT OF TCPIsGetReady() DIRECTLY TO TRUE. For example, "if(TCPIsGetReady(MySocket) == TRUE){...}" must be converted over to: "if(TCPIsGetReady(MySocket)){...}".

11.Changed TCPDiscard() return type from BOOL to void.

12.Removed TCP_NO_WAIT_FOR_ACK option. It was defaulted to disabled in the last two releases of the stack and is not needed with the new TCP module.

13.Updated DNS module to include two new required APIs: DNSBeginUsage() and DNSEndUsage(). These functions control a one bit ownership semaphore to allow multiple applications to use the DNS module in series. If invoked correctly, this will prevent unintended bugs resulting from two applications trying to use the DNS module at the same time. Old applications, such as those based around the GenericTCPClient.c example must be updated to use these functions.

14.Started using a new project structure and folders. You must use MPLAB 7.41 or higher (stack is tested on MPLAB 7.50) to use the default workspaces/projects, which include files using relative paths. This should improve compatibility with some future code libraries released by Microchip. StackTsk.h was broken into TCPIPConfig.h, HardwareProfile.h, and StackTsk.h. TCPIPConfig.h now includes all stack configuration options and HardwareProfile.h contains all hardware options. No macros need be globally defined in MPLAB project now. TCPIP.h is the only header applications must include now, for any/all modules used.

15. Combined ARP.c/ARP.h and ARPTsk.c/ARPTsk.h into a single file pair:

ARP.c/ARP.h. Applications built using a prior stack revision must remove all instances including "ARPTsk.h" and replace it with "ARP.h" instead. The ARP module is now simpler, more linear (easier to read), and being in one source file, allows the C compiler to optimize better.

16. Added PIC18F67J60_TEST_BOARD hardware profile to HardwareProfiles.h. This hardware profile is designed for 05-60091 (Rev 1), a development board that is not in production at this time.

17. Added DSPICDEMNET1 and DSPICDEMNET2 hardware profiles to HardwareProfiles.h for eventual support of the Microchip dsPICDEM.net 1 and dsPICDEM.net 2 demo boards. These two boards use the RTL8019AS Ethernet controller and a 24LC515 EEPROM. These changes are currently incomplete and these profiles cannot be used.

18. Began rewriting I2CEEPROM.c to support 16 bit CPUs, including the dsPIC30F6014 used on the dsPICDEM.net 1 and 2 demo boards. Note that work here is incomplete and cannot be used as a result -- see Known Problems section.

19. Partially updated RTL8019AS.c to support 16 bit CPUs, including the dsPIC30F6014 used on the dsPICDEM.net 1 and 2 demo board. Note that work here is incomplete and cannot be used as a result -- see Known Problems section.

20. Updated SNMP.c to use new typedefs in GenericTypeDefs.h. Also SNMP was tested in this version. SNMP.mib was updated some to better reflect current hardware.

21. Added AN870 SNMP callbacks to MainDemo.c (a feature that was missing in 3.xx releases). This code will get compiled when STACK_USE_SNMP_SERVER is defined in TCPIPConfig.h.

22. Removed all instances of MPFS_USE_PGRM for storing in internal FLASH program memory. Storage in internal program memory is now the default. Define MPFS_USE_EEPROM to override the default and store MPFS in an external EEPROM memory.

23. Decreased program memory needed for Announce.c module by about 180 bytes. Multiple inline calls to UDPPut() were removed.

24. UDP checksum checking logic has been improved. The UDP layer now avoids writing the pseudo header checksum in the RX buffer.

25. Swapped endianness of the returned checksum from CalcIPBufferChecksum(). Rewrote CalcIPBufferChecksum() in Helpers.c. This improves consistency.

26. Improved swapl() in Helpers.c.

27. Improved USART baud rate (SPBRG) calculation for PIC18s. Rounding is now done to choose the most optimal value and the code will automatically select high baud rate mode (BRGH=1) if possible. Additional improvements can be made if using a newer PIC18 with the 16 bit baud rate generator.
28. Added GenericTCPServer.c example file to complement GenericTCPClient.c. The server is enabled by defining STACK_USE_GENERIC_TCP_SERVER_EXAMPLE in TCPIPConfig.h.
29. Renamed STACK_USE_GENERIC_TCP_EXAMPLE definition to STACK_USE_GENERIC_TCP_CLIENT_EXAMPLE for consistency with new server example.
30. Defaulted MPFS.exe to generate binary MPFS images using 32 bit addressing. MPFS.h has been modified to also default to use 32 bit addressing of external EEPROM images. You must rebuild any old MPFS images and reprogram them if upgrading from a previous TCP/IP stack revision, which defaulted to use 16 bit addressing.
31. Updated MPFS.exe to #include "TCPIP.h" instead of "..\Headers\Compiler.h" in C files generated by the utility.
32. Added MPFSv2.exe PC utility for generating large MPFS images in program memory (ASM30 code) for C30 users. Previously, the C30 compiler placed a limit of less than 32KB of total MPFS size due to the PSV window size limitation on PIC24/dsPIC devices. To get around the limitation, use the new MPFSv2.exe utility to generate an .s file which can be included in your project instead of the .c file generated by the traditional MPFS.exe utility.

Fixes:

1. Fixed a bug in ARPProcess() which would incorrectly send an ARP response to an incorrect MAC & IP address if a TX buffer wasn't immediately available.
2. Fixed a TCP bug where TCPIsGetReady() would return TRUE even if no data was left in the received packet. Previously you had to call TCPGet() one last time and have it fail before TCPIsGetReady() would return FALSE.
3. Modified TCP state machine. Established connections will no longer automatically close if left idle for approximately 45 seconds. Note that your application needs to ensure that no sockets unintentionally get lost (For example: a server socket that received data only is established and the cable breaks while connected. In this case, the socket would never be detected as being disconnected since the server never attempts to transmit anything).
4. Stopped overclocking dsPIC33 and PIC24H devices. Previously PLLFBD was

incorrectly set to 39 instead of 38 to yield a resulting Fosc of 84MHz (42MIPS) instead of 80MHz (40MIPS) with the default Explorer 16 development board. Thanks go to Matt Watkins on the Microchip Ethernet Forum for pointing this error out.

5. Corrected a bug in IP.c where IPHeaderLen would not be properly initialized if a NON_MCHP_MAC was used (ex: RTL8019AS) and IPSetRxBuffer() was called. This bug did not affect ENC28J60 or PIC18F97J60 family support. Thanks go to Darren Rook for identifying this issue.

6. Updated checksum checking code in ENC28J60.c for latest silicon DMA checksum errata.

7. Declared TickCount in Tick.c/Tick.h as volatile and implemented an interrupt safe reading procedure in TickGet(). Since this multibyte variable is modified in the ISR and read in the mainline code, these changes are needed to prevent rare inconsistency bugs.

8. Fixed Announce.c so the unicast remoteNode of the requesting packet would be used rather than the remoteNode of the last received packet, which may not be correct when transmitting. Thanks go to Brett Caulton for identifying this issue.

9. Fixed a DHCP bug which would cause DHCP renewals to continually occur after only 60 seconds once the original lease expired. Thanks go to Brett Caulton for identifying this issue and fix.

10. Fixed a potential TCP socket leak in the FTP module. Previously FTPDataSocket would not be reliably initialized nor closed if the connection was killed forcefully (user killed application, cable disconnected while transferring, etc.).

Known Problems:

1. RTL8019AS MAC layer driver has not been updated for new TCP module. Users requiring RTL8019AS support should continue to use stack version 3.75.

2. I2CEEPROM.c has not been tested or completed. Continue to use I2CEEPROM.c from stack version 3.75 if this file is needed.

3. Telnet server module is still in development. No user authentication features are currently implemented. Some telnet clients may render the telnet server output incorrectly (in the wrong locations or wrong colors). Testing has only been done with the Microsoft Windows telnet.exe utility that comes Windows XP.

4. DHCP will continually send out DHCP Request packets when the lease expires and the original DHCP server that gave the lease is offline. The board will continue to

use the expired IP address until the DHCP server comes back online, at which point the lease will be renewed or a new discovery will occur. A new discovery should occur after timing out, instead. It is believed that this problem has always existed in previous stack revisions.

5. DHCP will continually send out DHCP Request packets when the lease expires and the original DHCP server that gave the lease does not include Option 54, the Server Identifier. A new discovery should occur after timing out. It is believed that this problem has always existed in previous stack revisions.

6. TFTPc module has not been tested with this version.

7. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).

v3.75 14 August 2006

Changes:

1. Added beta DNS client module (DNS.c). DHCP was also updated to obtain a DNS server address. Added AppConfig.PrimaryDNSServer IP address. Added STACK_USE_DNS configuration macro. To use the DNS client, call DNSResolve() with the server name, ex: DNSResolve("www.microchip.com"), and then periodically call DNSIsResolved() until it returns TRUE, ex: DNSIsResolved(&IPAddressDestination). Only one DNS resolution can be in progress at a time. Because the DNS client is a beta module, the API or code may change before being finalized. No formal DNS API documentation is available yet.

2. Added beta NetBIOS Name Service responder module (NBNS.c). Added AppConfig.NetBIOSName string. Added STACK_USE_NBNS configuration macro. Added MY_DEFAULT_HOST_NAME macro in StackTsk.h. Now, whenever a NetBIOS broadcast attempting to resolve AppConfig.NetBIOSName arrives, a response will be made. This form of name resolution only works on a single subnet. Off the subnet, manual registration in a DNS server or other means will be needed to allow the local Host Name to be recognized and translated to an IP address. The default NetBIOS name for the board is "MCHPBOARD". To test the NetBIOS Name Service module, try entering http://MCHPBOARD/ into your web browser instead of the board's IP address.

3. Added beta HTTP client module (GenericTCPClient.c). This module demonstrates how to make a TCP client application. To test this module, uncomment the STACK_USE_GENERIC_TCP_EXAMPLE macro in StackTsk.h, recompile, and then press the

BUTTON1 button while the stack is running. RemoteURL[] should be downloaded from ServerName[] and written to the UART. For the default values of ServerName[] and RemoteURL[], the HTML search page for "Microchip" will be fetched from "www.google.com" and written to the serial port. No formal documentation is available for this example yet.

4. Added Embedded Ethernet Device Discoverer PC project to aid in embedded product discovery when connected to a network and demonstrate how to write PC applications which can communicate with embedded devices. The source code for this device is included. It can be built using the Microsoft Visual C# 2005 Express Edition compiler. At the time of stack release, this 3rd party PC development tool can be downloaded at no cost from <http://msdn.microsoft.com/vstudio/express/>. If using only the Microchip Device Discoverer executable file without the Visual C# compiler, the .NET Framework 2.0 must be installed on the local PC. The application setup utility should allow dynamic downloading of this component if the target machine does not already have it installed.

5. Updated Announce.c to listen and respond to discovery requests sent to UDP port 30303 starting with the character 'D'. To test this functionality, use the Embedded Ethernet Device Discoverer on a PC connected to the same subnet.

6. Updated UART configuration menu to accommodate the new beta module configuration options (DNS server address, device host name).

7. Increased MPFS reserve block to 64 bytes from 32. Also, because the APP_CONFIG structure was updated, all current MPFS images and data stored in deployed EEPROMs needs to be updated.

8. Added a means to erase (invalidate) the onboard EEPROM using the BUTTON0 momentary switch (right-most switch on demo boards with multiple switches). To erase the EEPROM, hold down BUTTON0, RESET the board (press and release MCLR switch), and then continue to hold down BUTTON0 for an additional 4 seconds. If you press MCLR again, the EEPROM contents will now be invalid. If you press '0' on the UART, the same configuration that was read prior to invalidating the contents will be written back into the EEPROM. Invalidating the EEPROM allows the MY_DEFAULT_* constants to get loaded into a previously programmed EEPROM chip. Because of change #7, this procedure should be done for all currently programmed EEPROMs to prevent anomalous values from being read.

9. remoteNode in StackTsk.c was changed from private to global scope. Now external modules can reference the address of the last received packet. Announce.c uses this to send a unicast response to a broadcast discovery request.

10. All stack modules that can be disabled (DHCP.c, FTP.c, etc) now will no longer emit a compiler error if you have it in the project without defining the

appropriate macro (STACK_USE_DHCP, STACK_USE_FTP, etc). It will simply generate no machine code when compiled and the stack will not use that module. Make sure the proper macro is defined for each module that you wish to use.

11.Added SetRXHashTableEntry() to ENC28J60.c. This function can be used to set the appropriate bit in the Hash Table registers to join a particular multicast group.

12.Added Realtek RTL8019AS Ethernet controller support to the stack. MAC.c was renamed to RTL8019AS.c. This Ethernet controller is not recommended for new designs. RTL8019AS support was reintroduced to provide ongoing assistance to former Application designs implementing this chip. For new applications, use the Microchip ENC28J60 or PIC18F97J60 family of microcontrollers.

13.Added I2C EEPROM support for MPFS storage. In older 2.xx stack revisions, I2C EEPROM was supported by the XEEPROM.c file. This file has been renamed to I2CEEPROM.c. It is mutually exclusive with SPIEEPROM.c, and only one may be included in the project at a time.

14.Added new hardware definitions to Compiler.h. Pin mappings for the PICDEMNET and PIC18F97J60_TEST_BOARD boards have been added. FS_USB was also defined; however, it is untested and not recommended. See Compiler.h. The PIC18F97J60_TEST_BOARD is a non-production board that some Early Adopters of the PIC18F97J60 family parts have.

15.Changed type definitions for BYTE_VAL, WORD_VAL, DWORD_VAL, and moved the generic typedefs to GenericTypeDefs.h from StackTsk.h. This should improve compatibility with some future code libraries released by Microchip.

16.LCDBlocking.c module was modified to support 4-bit interfaces to LCD modules. The PICDEM.net board has the module wired using a 4-bit bus.

Fixes:

1. Fixed a serious MAC TXBuffer leak in TCP.c. Previously TCP.c would allocate a buffer for each socket in use, but under heavy traffic conditions (ex: user holds down F5 on web browser), the buffer handle might have been discarded before releasing the buffer. As a result all TCP connections would have lost the ability to send any application data after the TXBuffer pool ran out.
2. In the TCP_SYN_SENT TCP state, ACKs may only be received (as opposed to SYN+ACK packets) if the remote node thinks the connection is already open. A RST is now sent in response to an unexpected ACK, which may improve reconnection time when this (rare) condition occurs.
3. A bug was present in the UDP module where remote MAC addresses would be cached for each socket, even when UDPInit() or UDPClose() was called, or the microcontroller was reset. As a result, responses to incoming packets could have been sent to the

wrong MAC address. UDP Sockets are now properly initialized/closed.

4. Fixed a potential timing bug in LCDBlocking.c. For lower values of CLOCK_FREQ, insufficient delay time was given to the LCD module, potentially causing improper operation.

5. Changed PIC24F to default to the XT oscillator fuse rather than HS. The PIC24FJ128GA010 data sheet, rev. C reports that 8MHz should be used with XT mode, not HS mode like prior data sheets.

6. Added a couple of wait states to the Realtek RTL8019AS MAC layer module for NICPut() and NICGet(). Previously, the PICmicro could not operate above approximately 25MHz without losing communication with the RTL8019AS chip.

7. Updated PC based MPFS utility. When generating C files to be added to your MPLAB project, the include path to "Compiler.h" is now "..\Include\Compiler.h". The output file, ex: "MPFSImg.c" should be placed in the "Source" subfolder before compiling. For example, if you are in the main stack folder with the MPLAB projects, type: "mpfs /c WebPages Source\MPFSImg.c"

8. IP Gleaning will now get properly disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled. The stack will still respond to ping requests which have the wrong destination IP address, but a correct MAC address. However, the stack will continue to keep its statically defined IP address when DHCP/IP Gleaning are disabled and the ping arrives.

9. SPIEEPROM.c now saves and reconfigures the EEPROM_SPICON1 register (SSPCON1) before reading or writing to the SPI. After the read/write, it restores the saved state.

This allows the SPI bus to operate at different speeds, depending on what peripheral is being accessed if other devices share the bus and can support different speeds. In particular, this fixes the SPI @ 10.4MHz problem on the PICDEM.net 2 board when using the ENC28J60.

Known Problems:

1. DHCP will continually send out DHCP Request packets when the lease expires and the original DHCP server that gave the lease is offline. The board will continue to use the expired IP address until the DHCP server comes back online, at which point the lease will be renewed or a new discovery will occur. A new discovery should occur after timing out, instead. It is believed that this problem has always existed in previous stack revisions.

2. DHCP will continually send out DHCP Request packets when the lease expires and the original DHCP server that gave the lease does not include Option 54, the Server Identifier. A new discovery should occur after timing out. It is believed that this problem has always existed in previous stack revisions.

3. When an MPFS .c image file is added to a C30 project, a linking error reporting insufficient contiguous .const memory may occur when too much data is in the MPFS image (PSV window size limitation). Using the PSV window, 1 out of every 3 program memory bytes is wasted.
4. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.
5. SNMP, TFTPc modules have not been tested with this version.
6. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).
7. The C30 linker may misplace the __CONFIG2 section or disallow usage of MPFS images that are too big (add too much to the .const code section). The consequences of this are that the first configuration word at 0x157FC may not get set through code (must use the Configuration Bits dialog instead), and/or the project will not compile. This problem has been observed with C30 ver. 2.02 on the PIC24FJ128GA010 product. To work around this problem, the p24FJ128GA010.gld linker script has been modified. Specifically, line 68 has been commented out, which causes the linker to place all .text sections after placing all absolute sections. SSR 25966 in the C30 2.02 release notes may be related.
8. It is observed with the Realtek RTL8019AS Ethernet controller and the demo AJAX web page which self refreshes rapidly, that occasional HTTP GET requests sent by the computer do not get received by the HTTP server. This is believed to be a RTL8019AS MAC layer bug. The TCP protocol handles the packet loss, but application performance suffers while waiting for the TCP retransmission. This problem is not observed with ENC28J60.c or ETH97J60.c MAC layers.
9. The HI-TECH compiler version 9.50PL1 crashes when compiling LCDBlocking.c with 4 bit mode (PICDEMNET) and using a warning level of -3 or higher. To work around the problem, the HI TECH projects were set to use warning level -4.

Guiding Notes:

1. To use the stack on a classic PICDEM.net demo board with the Realtek Ethernet controller, a PIC18F452 processor, and Microchip C18:

- Use the C18EEPROM MPLAB project
- Change the processor in the MPLAB IDE
- Change linker script to "18f452i.lkr" in the MPLAB project. Use the one provided in the Linker subfolder, it has been modified to make more RAM available.
- Update the hardware definitions macro. Click on Project -> Build Options... -> Project -> MPLAB C18 -> Add PICDEMNET, remove HPC_EXPLORER)

- Remove ENC28J60.c from the project
- Remove SPIEEPROM.c from the project
- Add RTL8019AS.c to the project
- Add I2CEEPROM.c to the project
- Enable all compiler optimizations (Project -> Build Options... -> Project -> MPLAB C18 -> Categories Optimization -> Enable all)

v3.60 12 July 2006

General Information:

This stack version is being publicly released, so the following changes are with respect to the prior public stack release (v3.02). Interim stack changes for version 3.16 and 3.50 are documented below for those using non-public releases, but can be ignored by most people.

Troubleshooting notes:

1. If you have an Ethernet PICtail revision 2.1 and are having reliability issues when viewing the fast-refresh demo web page, you may need to install resistors in series with the ENC28J60 SI, nCS, and SCK pins. The recommended value is 100 to 200 ohms. This will reduce signal undershoot caused by long traces (parasitic inductance), which can violate the absolute maximum electrical specs and cause SPI data corruption. The HPC Explorer Rev 5 has fairly long traces to the PICtail connector.
2. Enabling C30 2.02 compiler optimizations on the dsPIC33FJ256GP710, PIC24HJ256GP610 ES chips may produce unreliable code.
3. When changing a C30 project to a PIC24H or dsPIC33F processor on the Explorer 16 demo board, the JTAG configuration fuse should be disabled to free the I/O pins associated with it. JTAG is enabled by default.
4. This stack release was tested using MPLAB 7.40, C18 version 3.03, C30 version 2.02, and HI TECH PICC18 version 9.50PL1.
5. When using the Ethernet PICtail board and HPC Explorer demo boards, make sure to plug the power into the Ethernet PICtail and not the HPC Explorer. The HPC Explorer's power regulator cannot provide enough current.

Changes:

1. Source files have been split into separate directories. To compile old applications with this new stack, application source files may need to be updated to include the proper path to the stack header files.

2. New MPLAB projects have been created:

-C18EEPROM: Equivalent to the previously named "mpnicee" project. Designed for PIC18's using the C18 compiler. Web page content, board's IP address, MAC address, DHCP enabled state, etc. is stored in an external SPI EEPROM (25LC256 on demo boards). FTP Server demo is included.

-C30EEPROM: New supporting PIC24 and dsPIC controllers using the C30 compiler. Similar to C18EEPROM.

-C18ProgramMem: Equivalent to the previously named "mpnicpg" project. Web page content stored in internal FLASH program memory. Board's IP address, MAC address, DHCP enabled state, etc. is stored only in RAM and defaults are loaded from MY_DEFAULT_* constants in StackTsk.h. FTP Server demo is not included. Web pages cannot be updated remotely.

-C30ProgramMem: New supporting PIC24 and dsPIC controllers using the C30 compiler. Similar to C18ProgramMem.

-HTC18EEPROM: Equivalent to the previously named "htnicee" project. Designed for PIC18's using the HI TECH PICC18 compiler. Similar to C18EEPROM.

-HTC18ProgramMem: Equivalent to the previously named "htnicpg" project. Designed for PIC18's using the HI TECH PICC18 compiler. Similar to C18ProgramMem.

3. Created hardware definitions (pins, interrupt flags, special registers, etc) in Compiler.h for easy changing of hardware. Four demo board combinations are supported out-of-box now:

-EXPLORER_16: Explorer 16 motherboard + Ethernet PICtail Plus daughter card. Tested with dsPIC33FJ256GP710, PIC24HJ256GP610, and PIC24F128GA010 ES PIMs.

-HPC_EXPLORER: PICDEM HPC Explorer motherboard + Ethernet PICtail daughter card. Tested with PIC18F8722 onboard and PIC18F87J10 PIM.

-DSPICDEM11: dsPICDEM 1.1 motherboard + Ethernet PICtail daughter card (manually air wired). See Compiler.h for proper pins to air wire. Tested with dsPIC30F6014A PIM.

-PICDEMNET2: PICDEM.net 2 motherboard (PIC18F97J60)

Change boards by changing the defined macro (Project -> Build Options... -> Project -> MPLAB Cxx -> Add macro). When moving to custom hardware, add an appropriate profile to Compiler.h. YOUR_BOARD is present as a placeholder.

4. Added Ethernet PICtail Plus schematic (reference ENC28J60 daughter card design for Explorer 16 demo board). These boards have a Microchip part number of AC164123.

5. Latest ENC28J60 rev. B5 errata workarounds added. The code checks the EREVID

register and implements the appropriate workarounds as needed for the silicon revision, so rev. B1, B4, and B5 are all supported in this stack release.

6. Significantly revised demonstration web page content in WebPages folder to use AJAX technology. Using asynchronous JavaScript code executing in the web browser, the status sections of the page are updated rapidly from the web server without doing a full page refresh. As a result, a virtually real time update of the potentiometer and button values can be displayed. Due to the constant use of new TCP sockets, multiple simultaneous users are not recommended. See the Index.cgi file for a simple static method of retrieving dynamic variables from the HTTP server.

7. Changed IP Gleaning procedure. Now, if DHCP is enabled, the DHCP module will continue to look for a new IP address/renew existing IP address if the IP address is configured using IP Gleaning. Previously, the DHCP module would be disabled once a successful ICMP packet was received and used to configure the IP address.

8. MAX_RETRY_COUNTS is 3 (previously it was 3, but an interim release changed it to 5).

9. Updated TCP state machine. It now includes the TCP_FIN_WAIT_2 state. Some other changes were made to handle errors more robustly.

10. AN0String and AN1String now return all characters excluding the null terminator when the HTTP server calls HTTPGetVar (except when the string is 0 length). Previously, the null terminator was returned as well.

11. Dynamic pages (ie: .cgi files) are now served with an expired HTTP header to prevent browser caching and allow more dynamic content to be displayed.

12. Support for the HI TECH PICC18 compiler has changed. Special Function Register bits and other definitions have changed substantially from the previous HI TECH PICC18 projects in TCP/IP stack version 3.02 and earlier. The C18/C30 SFR and SFRbits naming conventions are now used and special remapping macros in Compiler.h are used to maintain a consistent syntax. The HI TECH PICC18 projects were tested with compiler version 9.50PL1 on the HPC Explorer board (PIC18F8722).

13. FTP client hash printing has been added to the FTP server. Now, whenever a chunk of data is successfully uploaded to the device, a '#' character will appear on the FTP client screen. The numbers of bytes each '#' represents is variable.

14. To improve maintainability, built in support for the "Compatible" A/D converter present on older PIC18 parts (ex: PIC18F452) has been removed.

15. Removed old LCD code originally provided for the PICDEM.net demo board.

16. Added LCDBlocking.c and LCDBlocking.h, which implement simple routines for writing to the LCD module on the Explorer 16 and PICDEM.net 2 development boards. The LCD on the dsPICDEM 1.1 board is not supported. The stack version and IP address are shown on the LCD on power up.

17. UART functions in MainDemo.c were replaced with C18 and C30 peripheral library functions. However, because the UART peripheral libraries are not being updated for newer silicon devices, the code was copied into UART.c and is compiled with the stack.

18. Multiple TX buffer support has been implemented. Most stack layers have been touched. ENC28J60.c has the most extensive changes. Each socket may use only one TX buffer.

19. Implemented TCP retransmission support regardless of if TCP_NO_WAIT_FOR_ACK is defined or not.

20. TCP_NO_WAIT_FOR_ACK in StackTsk.h has been undefined by default. This should increase default TCP connection robustness. Packets sent from the stack to the remote node will now be detected and retransmitted if lost or corrupted.

21. All TCP packets are now retransmitted immediately after being initially transmitted when TCP_NO_WAIT_FOR_ACK is undefined. This improves throughput greatly when communicating with systems which wait a long time before transmitting ACKs. TCP/IP stacks, such as that used by Microsoft Windows, implement the TCP Delayed Acknowledgement algorithm, which is why this retransmission is necessary for high performance. The double transmission feature can be disabled in the Microchip TCP/IP stack by defining "DEBUG" either in the TCP.c file or the project compiler macros section. Using DEBUG mode can be useful when trying to look for errors using Ethreal [<http://www.ethereal/>].

22. Lowered TCP_START_TIMEOUT_VAL from 60 seconds to 3 seconds. 60 seconds is an unreasonably long timeout for modern day network speeds.

23. Native support for the SLIP module has been dropped.

Fixes:

1. A new IP address obtained via IP Gleaning will now update the LCD (if present), invoke the Announce module (for MCHPDetect.exe), and output the new address out the RS232 port.

2. DHCP client will now correctly use the first DHCP offer received when connected to a network running multiple DHCP servers. Previously, the board would get no IP address when attached to a network with multiple DHCP servers (unless the DHCP request was transmitted before a second DHCP offer was received -- a relatively rare event). Additionally, DHCPLeaseTime does not get reset to 60 seconds or the value stored in the last DHCP packet received prior to receiving the ACK.

3. UDPProces() will now correctly process received UDP packets that have a 0x0000 checksum field. The UDP protocol specifies that 0x0000 means the checksum is disabled. Packets with a 0x0000 checksum were previously thrown away unless the

calculated checksum also happened to be 0x0000.

4. The `TCPIsPutReady()` function will now honor the remote node's TCP window size. In other words, if the remote application pauses or cannot handle the incoming data rate, the TCP flow control feature will correctly function. Previously, if the remote node ran out of incoming buffer memory, the TCP layer would still allow more data to be transmitted. This would result in the loss or corruption of application data, with a potentially broken connection. The change requires 2 more bytes of RAM per TCP socket (TCB array).

Known Problems:

1. On PICDEM.net 2 board ENC28J60 and 25LC256 EEPROM share the same SPI1 module. At 3.3V, the 25LC256 is only rated to 5MHz SPI clock, but the code is setting it to 10.4MHz because the `MACInit()` function reconfigures the same SPI1 module.
2. DHCP will continually send out DHCP Request packets when the lease expires and the original DHCP server that gave the lease is offline. The board will continue to use the expired IP address until the DHCP server comes back online, at which point the lease will be renewed or a new discovery will occur. A new discovery should occur after timing out, instead. It is believe that this problem has always existed in previous stack revisions.
3. DHCP will continually send out DHCP Request packets when the lease expires and the original DHCP server that gave the lease does not include Option 54, the Server Identifier. A new discovery should occur after timing out. It is believe that this problem has always existed in previous stack revisions.
4. The MPFS utility has not been updated. When creating a .c image file, the include path for the `Compiler.h` file will be incorrect and need to be manually updated to `"..\Include\Compiler.h"`.
5. When an MPFS .c image file is added to a C30 project, a linking error reporting insufficient contiguous .const memory may occur when too much data is in the MPFS image (PSV window size limitation). Using the PSV window, 1 out of every 3 program memory bytes is wasted.
6. `MACSetPMFilter()`, `MACDisablePMFilter()`, and `MACCopyRxToTx()` have not been tested and possibly do not work.
7. SNMP, TFTPc modules have not been tested with this version.
8. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).
9. IP Gleaning may not get disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled.

10. The C30 linker may misplace the `__CONFIG2` section or disallow usage of MPFS images that are too big (add too much to the `.const` code section). The consequences of this are that the first configuration word at `0x157FC` may not get set through code (must use the Configuration Bits dialog instead), and/or the project will not compile. This problem has been observed with C30 ver. 2.02 on the PIC24FJ128GA010 product. To work around this problem, the `p24FJ128GA010.gld` linker script has been modified. Specifically, line 68 has been commented out, which causes the linker to place all `.text` sections after placing all absolute sections. SSR 25966 in the C30 2.02 release notes may be related.

Guiding Notes:

1. To change processors using a C18* project:

- Change the processor in the MPLAB IDE
- Change linker script (ex: `18f87j10i.lkr`) in the MPLAB project. Use `*i.lkr` if the ICD2 is going to be used to debug with.
- Update the hardware definitions in `Compiler.h` or change your demo board selection macro. (Project -> Build Options... -> Project -> MPLAB Cxx -> `PICDEMNET2`, etc)

2. To change processors using a HTC18* project:

- Change the processor in the MPLAB IDE
- Update the hardware definitions in `Compiler.h` or change your demo board selection macro. (Project -> Build Options... -> Project -> MPLAB Cxx -> `PICDEMNET2`, etc)

3. To change processors using a C30* project:

- Change the processor in the MPLAB IDE
- Change linker script (ex: `p33FJ256GP710.gld`) in the MPLAB project.
- Update the hardware definitions in `Compiler.h` or change your demo board selection macro. (Project -> Build Options... -> Project -> MPLAB Cxx -> `DSPICDEM11`, etc)
- Disable JTAG configuration fuse, if enabled

4. When using the PICDEM.net 2 board, to write code targeting the PIC18F97J60 family Ethernet module:

- Remove `ENC28J60.c` from the project
- Add `ETH97J60.c` to the project
- Plug the Ethernet cable into the left-most RJ45 jack (next to LCD)

5. When using the PICDEM.net 2 board, to write code targeting the ENC28J60 Ethernet device:

- Make sure `ENC28J60.c` is in the project

- Make sure that ETH97J60.c is not in the project
 - Plug the Ethernet cable into the right-most RJ45 jack (next to board edge)
6. When using the PICDEM.net 2 board, to write code targeting an Ethernet PICtail module (ENC28J60):
- Make sure ENC28J60.c is in the project
 - Make sure that ETH97J60.c is not in the project
 - Make sure that the Ethernet PICtail J9 jumper is in the 2-3 position (default).
 - Properly update the hardware profile in Compiler.h. ENC_CS_TRIS and ENC_CS_IO need to be changed from D3 to B3.
 - Plug the Ethernet cable into the PICtail
 - Plug power into the PICDEM.net 2 board
7. When using the Explorer 16 and Ethernet PICtail Plus demo boards, make sure to mate the PICtail to the motherboard using the topmost socket position, leaving the cable hanging over prototyping area. If SPI2 is desired, the PICtail should have the same orientation but be installed in the middle slot. Using SPI2, the hardware profile will need to be updated in Compiler.h.

v3.50 13 April 2006

Changes:

1. Improved dsPIC33F and PIC24H support. UART functions are included now instead of precompiled object files for the PIC24F. The 12-bit A/D converter is now shown in use on the demo web content. When changing a C30 project to a PIC24H or dsPIC33F processor on the Explorer 16 demo board, the JTAG configuration fuse should be disabled to free the I/O pins associated with it. JTAG is enabled by default.
2. Added LCDBlocking.c and LCDBlocking.h, which implement simple routines for writing to the LCD module on the Explorer 16 development board. The stack version and IP address are shown on the LCD on power up.
3. Added "C18ProgramMem" and "C30ProgramMem" MPLAB projects for MPFS storage (web page content) on on-chip program memory. These projects are equivalent to the previously named "mpnicpg" project in prior stack releases.
4. Multiple TX buffer support has been implemented. Most stack layers have been touched. ENC28J60.c has the most extensive changes. Each socket may use only one TX buffer.
5. Implemented TCP retransmission support when TCP_NO_WAIT_FOR_ACK is undefined.
6. TCP_NO_WAIT_FOR_ACK in StackTsk.h has been undefined by default. This should increase default TCP connection robustness.

7. All TCP packets are now retransmitted immediately after being initially transmitted when TCP_NO_WAIT_FOR_ACK is undefined. This improves throughput greatly when communicating with systems which wait a long time before transmitting ACKs.
8. Lowered TCP_START_TIMEOUT_VAL from 60 seconds to 3 seconds.
9. Increased MAX_RETRY_COUNTS from 3 to 5 times.
10. The example HTTP server now returns a content expiration date which has already past. This prevents web browser caching and allows more dynamic content to be displayed.
11. Added WebPages_JScript folder, with new web pages that support dynamic page updates without a full page reload. A tiny page of dynamic variables is returned by the web server and Javascript executing on the target web browser changes DOM elements as needed. Button S5 (RA7) on the Explorer 16 demo board and S1 (RB0) on the HPC Explorer demo board changes the page color scheme. The rapid dynamic updates do not work on some web browsers (Internet Explorer works, Firefox does not).

Known Problems:

1. MPFS utility has not been updated. When creating a .c image file, the include path for the compiler.h file will be incorrect and need to be manually updated.
2. When an MPFS .c image file is added to a C30 project, a linking error reporting insufficient contiguous .const memory may occur (PSV window size limitation).
3. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.
4. SNMP, TFTPc, SLIP modules have not been tested with this version.
5. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).
6. IP Gleaning may not get disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled.
7. The IP address being outputted out the RS232 port and through the Announce module does not happen when the IP address is configured using IP Gleaning.
8. On the PIC24F with C30 compiler optimizations enabled (such as Option 3, maximum speed), the project may not work. The PIC24F headers that come with C30 ver. 2.01 declare several SFRs without using the volatile keyword.
9. dsPIC30 support is incomplete. Currently PIC18, PIC24F, PIC24H, and dsPIC33F processors are supported.

v3.16.00: 06 March 2006

Changes:

1. Added unified support for both the Microchip C18 and C30 compilers. The intention is to allow one code base to be compiled for any PIC18, PIC24F/H, dsPIC30, or dsPIC33 product (with adequate memory). See the "Tested Using" section for what is known to work.
2. To improve maintainability, support for the HI-TECH PICC18 compiler has been dropped.
3. New project workspaces have been created, "C30EEPROM.mcw" and "C18EEPROM.mcw". C18EEPROM.mcw is equivalent to the previously named "mpnicee.mcw." C30EEPROM is intended to be used for PIC24 and dsPIC 16-bit controllers.
4. Source files have been split into separate directories.
5. Latest ENC28J60 rev. B5 errata workarounds added. The code checks the EREVID register and implements the appropriate workarounds as needed for the silicon revision, so rev. B1, B4, and B5 are all supported in this stack release.
6. Removed old LCD code originally provided for the PICDEM.net demo board.
7. To improve maintainability, built in support for the "Compatable" A/D converter present on older PIC18 parts (ex: PIC18F452) has been removed.
8. UART functions in MainDemo.c were replaced with C18 and C30 peripheral library functions.

Tested Using:

1. Software:

-MPLAB version 7.31.01

-C18 version 3.02

-C30 version 2.01

2. Hardware:

-PICDEM HPC Explorer rev. 4 (PIC18F8722) + Ethernet PICtail Daughter Board (ENC28J60 B1)

-Explorer 16 rev. 4 (PIC24FJ128GA010 ES and dsPIC33FJ256GP710 ES) + Ethernet PICtail+ Daughter card (ENC28J60 B1).

3. Notes:

-MPLAB 7.31.01 is a development build. The publicly available version 7.31 should work fine, with the exception of being unable to program dsPIC33 and PIC24H parts with the ICD 2.

-No dsPIC30 or PIC24H parts have been tested yet.

Known Problems:

1. MPFS utility has not been updated. When creating a .c image file, the include path for the compiler.h file will be incorrect and need to be manually updated.
2. When an MPFS .c image file is added to a C30 project, a linking error reporting insufficient contiguous .const memory may occur.
3. On the PIC24FJ128GA010, it is observed that some inbound packets are lost from time to time with no anticipated reason.
4. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.
5. SNMP, TFTPc, SLIP modules have not been tested with this version.
6. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).
7. IP Gleaning may not get disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled.
8. The IP address being outputted out the RS232 port and through the Announce module does not happen when the IP address is configured using IP Gleaning.
9. Multiple TX buffer support is not fully implemented in the MAC layer, ENC28J60.c. Stack behavior when TCP_NO_WAIT_FOR_ACK is undefined may be unexpected.

v3.02.00: 20 Feb 2006

Fixes:

1. Changed TXSTART in ENC28J60.c to stop wasting a byte.
2. Changed RXSTOP in ENC28J60.c to always be an odd value to properly implement an ENC28J60 silicon errata workaround.
3. Changed initialization of ERXRDP in MACInit() to agree with the current errata.

Changes:

1. Licence agreement
2. Schematics and other board files to the Ethernet PICtail Daughter Board have been updated to revision 5. Of significant note, the nRESET pin has been freed and 200 ohm resistors were added to the ENC28J60 SI, nCS, and SCK pins. The added resistors reduce undershoot caused by stray trace inductance and strong host output drivers.

Known Problems:

1. Testing on the PICDEM.net demo board with the Realtek RTL8019AS Ethernet controller

has not been done. Moving to the HPC Explorer demo board has resulted in pinout and other hardware changes.

2. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.

3. SNMP, TFTPc, LCD, SLIP modules have not been tested with this version.

4. The stack may behave incorrectly if compiled using the Hitech compiler with a high optimizations setting.

5. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).

6. IP Gleaning may not get disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled.

7. The IP address being outputted out the RS232 port and through the Announce module does not happen when the IP address is configured using IP Gleaning.

8. Multiple TX buffer support is not fully implemented in the MAC layer, ENC28J60.c. Stack behavior when TCP_NO_WAIT_FOR_ACK is undefined may be unexpected.

v3.01.00: 18 Jan 2006

Fixes:

1. Implemented latest ENC28J60 silicon errata workarounds.
2. Fixed a bug in TCP.c and UDP.c which would incorrectly write the packet checksum into the RX buffer incorrectly when the checksum field was exactly spanning the RX wraparound boundary in the ENC28J60. This problem would have caused packets to be discarded in rare circumstances

Known Problems:

1. Testing on the PICDEM.net demo board with the Realtek RTL8019AS Ethernet controller has not been done. Moving to the HPC Explorer demo board has resulted in pinout and other hardware changes.
2. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.
3. SNMP, TFTPc, LCD, SLIP modules have not been tested with this version.
4. The stack may behave incorrectly if compiled using the Hitech compiler with a high optimizations setting.
5. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed

on power up).

6. IP Gleaning may not get disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled.

7. The IP address being outputted out the RS232 port and through the Announce module does not happen when the IP address is configured using IP Gleaning.

8. Multiple TX buffer support is not fully implemented in the MAC layer, ENC28J60.c. Stack behavior when TCP_NO_WAIT_FOR_ACK is defined may be unexpected.

v3.00.00: 16 Jan 2006

Changes:

1. The stack now targets the PICDEM HPC Explorer demo board (PIC18F8722 MCU) with an attached Ethernet PICtail Daughter Board (with the Microchip ENC28J60 Ethernet controller).

2. IP Gleaning is no longer enabled (STACK_USE_IP_GLEANING is not defined) by any of the default project files.

3. The IP address, whenever it changes, is outputted out the RS232 serial port in human readable form. Any terminal program, such as HyperTerminal can be used to read it. This allows the IP address to be easily determined when DHCP is used.

The serial port defaults to 19200 baud when CLOCK_FREQ in Compiler.h is properly defined.

Additions:

1. Microchip ENC28J60 Ethernet controller support. Support is included through the ENC28J60.c and ENC28J60.h files. Various other files were modified to take advantage of ENC28J60 specific features, like the hardware DMA/IP checksum engine.

This new MAC driver incorporates several new functions which can be called from any layer above the MAC. The functions are:

MACSetDuplex()

MACPowerDown()

MACPowerUp()

MACSetPMFilter()

MACDisablePMFilter()

CalcIPBufferChecksum()

MACCalcRxChecksum()

MACCalcTxChecksum()

MACCopyRxToTx()

See the ENC28J60.c file comments for function descriptions. The ENC28J60.c file also incorporates TestMemory() which can do a power on self test of various hardware functions. TestMemory() is included and used when MAC_POWER_ON_TEST is defined in StackTsk.h. It is undefined by default. Defining it will require some program memory.

2. Announce module. Announce.c and announce.h have been added. When included in the project, STACK_USE_ANNOUNCE must be defined. This module will broadcast a UDP message to port 30303 containing the local MAC address whenever the local IP address changes. This addition is intended to facilitate device discovery on DHCP enabled networks and eliminate the need for an RS232 connection if board reconfiguration is not needed. To retrieve the UDP message on your computer, use the new MCHPDetect.exe program included in the \MCHPDetect subfolder.

3. The spieeprom.c file was added to support SPI EEPROM chips for MPFS storage. ENC28J60.c and spieeprom.c may both be included and they will share the same SPI module.

Improvements:

1. Renamed files/edited files so that the HI-TECH compiler won't raise messages stating that include files were spelled wrong.
2. Moved MAX_ICMP_DATA_LEN from StackTsk.c to ICMP.h file for easier maintenance.
3. Corrected STACK_USE_SIIP typo in dhcp.c file - Thanks to Gisle J.B.
4. Implemented UDP checksum logic in UDPProcess() in UDP.c file.
5. Renamed CalcTCPChecksum() in tcp.c file to CalcIPBufferChecksum().
6. Moved CalcIPBufferChecksum() to helpers.c to reuse it for UDP checksum calculation.
7. Modified UDPProcess() in UDP.c and TCPProcess() in TCP.c to include localIP as third new parameter. This makes pseudo header checksum calculation correct in both functions. StackTsk.h, UDP.h and TCP.h files were also modified to reflect these changes.
8. Modified TCP.C file to include compile-time check of STACK_USE_TCP define. If it is not defined, an error will be displayed.
9. Removed an unnecessary call to MACDiscardRx() when an IP packet is received but fails version, options length, or header checksum tests.
10. Changed LCD code to be compile time removable by undefining USE_LCD.

Fixes:

1. IPHeaderLen in IP.c is initialized properly now when IPGetHeader() is called.
2. Under some circumstances, HandleTCPseg() would acknowledge, but throw valid received TCP packets away, resulting in loss of application data. An invalid comparison in

HandleTCPSeg() has been fixed to prevent this situation from occurring.

*** Thanks go to Richard Shelquist for identifying this problem.

3. Fixed StackTsk.c file so that if a static IP address is used and the LINK is removed, the node IP address is not cleared.
4. Invalid ICMP echo replies are no longer generated for echo requests with a data length of 33 (one more than the configured maximum).
5. Changed MAX_OPTIONS_LEN from 20 to 40. The maximum IP options length is now in agreement with the IP RFC.
6. Changed IPSetRxBuffer() from a macro to a function. The function takes into account any options which may be present in the header of received IP packets. Previously, possible options were not taken into account when calculating the offset.

Known Problems:

1. Testing on the PICDEM.net demo board with the Realtek RTL8019AS Ethernet controller has not been done. Moving to the HPC Explorer demo board has resulted in pinout and other hardware changes.
2. Sometimes when the FTP sever is used, an attempt to put a file is unsuccessful. The problem may be caused when an HTTP request to GET a file is made at the wrong time.
3. MACSetPMFilter(), MACDisablePMFilter(), and MACCopyRxToTx() have not been tested and possibly do not work.
4. SNMP, TFTPc, LCD, SLIP modules have not been tested with this version.
5. The stack may behave incorrectly if compiled using the Hitech compiler with a high optimizations setting.
6. Serial numbers >32K will be displayed on the serial port as a negative value when C18 is used and the board is placed in configuration mode (RB0 button is depressed on power up).
7. IP Gleaning may not get disabled when, through the RS232 configuration application, DHCP and IP Gleaning are disabled.
8. The IP address being outputted out the RS232 port and through the Announce module does not happen when the IP address is configured using IP Gleaning.
9. Multiple TX buffer support is not fully implemented in the MAC layer, ENC28J60.c. Stack behavior when TCP_NO_WAIT_FOR_ACK is defined may be unexpected.

v2.20.04.01: 9/24/03

1. Recreated MPLAB projects to avoid problems when source is not at \MCHPStack location.

v2.20.04: 9/5/03

Fixes:

1. Modified DHCPReset() in DHCP.c to not reset DHCP state machine if it was previously disabled using DHCPDisable(). This would make sure that if DHCP module was enabled and application had run-time disabled DHCP and network cable is disconnected, stack will not clear its IP address.

2. Rebuilt mib2bib.exe file with static library options. This fixes problem where one tries to execute this exe, an error occurs about missing DLLs.

v2.20.03:

Improvements:

1. When DHCP is enabled, LINK is monitored and IP address is reset on disconnect. New IP configuration is obtained on LINK reconnect. - For RealTek only.
Modified DHCP.c to add DHCPReset()
Modified MAC.c to add MACIsLinked()
Modified StackTsk.h to add BYTE_VAL def.

Changes:

1. Modified SMSC91c111.c to add empty MACIsLinked() - will be populated in next rev.

Bug Fixes:

1. Corrected DHCP logic to accept first DHCP offer instead of second response.
2. Corrected DHCP logic to check for chaddr in DHCP offer and accept one that matches with local MAC address. This will fix problem where if multiple nodes were on bus and all requested DHCP address, all would accept response from one server instead of verifying who was intended node.
3. Fixed UDPClose() in UDP.c to use INVALID_UDP_PORT instead of INVALID_UDP_SOCKET because of which a closed socket would not be scanned correctly.
4. Modified UDP.h to use long constant designators for INVALID_UDP_OPRT to explicitly state that it is a long.

v2.20.02:

Beta version containing TFTP client module.

Addition:

1. TFTP Client module - See TFTPc.* and TFTPcDemo.c for more information.

See MpTFTPcDemo and HtTFTPcDemo projects for build information.

Bug Fix:

1. UDP_IsGetReady() was modified to overcome compiler rule where only 8-bit value was used to evaluate non-zero condition.

2. ARPResolve() in ARPTsk was fixed to clear Cache.IPAddr value.

v2.20.01:

Bug fix:

1. Fixed SMSC91C111.c where MACInit() would hang if ethernet link is not detected.

v2.20:

Bug Fixes:

1. General - Removed most of harmless warnings.

2. C18Cfg.asm - Fixed "include" instead of "define".

3. DHCP.c - Increased DHCP_TIMEOUT_VAL to 2 seconds.

Fixed problem where UDP active socket was not set before calling UDP functions

in SM_DHCP_BROADCAST state.

4. MAC.c - Fixed MAC_IsTxReady() where under heavy traffic it would always return FALSE.

This fixes bug where all high level applications would stop transmitting.

5. TCP.c - Enabled portion of code that performs timeout logic even if TCP_NO_WAIT_ACK

is defined. This fixes bug where occasionally, tcp applications such as HTTP server would stop working after few hours.

6. UDP.c - Fixed UDPGet() where it would return FALSE on last good byte.
Fixed UDPPProcess() where it was calculating incorrect length.

Added bFirstRead flag with UDP sockets similar to TCP sockets
so that whenever first UDP byte is read, MAC read pointer will be
reset to beginning of correct packet.

This change fixes problem where if one transmits a packet while
UDP packet is pending in a socket, next get to pending UDP socket would
return wrong data. (This is apparent only when there is heavy network
traffic)

Known Issues:

1. HiTech v8.20 PL4 with all optimization enabled may not work properly.
2. C18 "Static" and "Auto" mode may not be used - there are too many local variables to fit
in standard stack of 256 bytes. One may modify linker script file to avoid this
limitation.

Improvements:

1. Modified TICK def. in Tick.h to unsigned long to support 32-bit wide SNMP tick.
2. Added SNMP Module (SNMP.c)
3. Added Two new demo projects - DemoSNMPApp and HtDemoSNMPApp.
4. Created MPLAB 6.X projects for different demo configurations.
5. MAC.c - Added MACGetTxOffset().
6. MPFS.c - Added MPFSSeek(), MPFSTell().
7. MPFSImg.*- Rebuilt to reflect v2.20, footprint changes etc.
8. StackTsk.h- Enhanced WORD_VAL, DWORD_VAL defs.
Added STACK_USE_SNMP and related compile-time checks.
9. UDP.h - Added UDPSetTx and UDPSetRx macros.
Moved UDP_SOCKET_INFO structure to header file.
10. WebSrvr.c- Modified MCHPStack version message and added DATE info to BoardSetup
menu.
11. Added support for SMSC LAN91C111 10/100 Non-PCI ethernet controller
Use "SMSC91C111.C" instead of MAC.c.
"mpnicee_smsc" is a sample project that uses PIC18F8720 and SMSC NIC.
"MasterDemo.c" is a main source file for above project that includes
all modules - must use device with more than 32KB of memory.

v2.11:

Bug Fixes:

1. Fixed dhcp.c to make it work with new C18 startup code.

Improvements:

1. Modified webservr.c DownloadMPFS() to make use of compiler allocated XMODEM data block rather than use fixed address block starting at 0x400.

v2.10: 7/9/02

Bug Fixes:

1. Fixed HTTP Server bug where a form submission with empty parameter value would not parse correctly.

v2.0: 5/22/02

New Modules:

1. Added UDP, DHCP, FTP and IP Gleaning
2. Added PICDEM.net LCD support
3. Added board setup through RS-232.

Improvements:

1. Optimized serial EEPROM access routines in terms of speed and size
(Replaced ee256.* files with eeprom*.h)
2. Improved board setup through RS-232.

Known Issues:

1. LCD may not display properly on MCLR only.

Workaround: 1. Debug XLCDInit() routine in "xlcdlh"

2. Always do POR reset.

2. SLIP connection is not very robust.

Workaround: None at this time.

3. Hi-Tech Compiler:

1. Aggressive optimization breaks the functionality.

Workaround: Apply optimization listed in each source file comment header.

2. In order to use V8.12, you will need to remove "FTP Server" from Ht*.pjt.

You will also need to disable all optimizations.

4. C18 Compiler: 1. Static model does not compile.

Workaround: None at this time.

2. Overlay model breaks the functionality.

Workaround: None at this time.

3. All modules does not fit in 32KB memory.

Workaround: 1. None at this time.

2. Sample project disables some modules.

New Files:

```
=====
=====
```

File Purpose

```
=====
=====
```

1. delay.* Provides CLOCK_FREQ dependent delay routines.

2. dhcp.* DHCP client support

3. ftp.* FTP server

4. udp.* UDP socket support

5. xeprom.* Improved ee256.* and renamed.

6. xlcd.* External LCD support.

7. version.log To track changes and history.

Changes:

=====

=====

File Change

To-do for v1.0 stack applications

=====

=====

1. arptsk.c 1. Fixed STACK_CLIENT_MODE compile errors.

None

2. Modified ARP_IsResolved() to support IP Gleaning

None

2. c18cfg.asm 1. Added PIC18F452 configuration

None

2. Fixed "include" errors.

None

3. compiler.h 1. Included "stdlib.h" in both C18 and Hi-Tech compilers.

None

2. Moved CLOCK_FREQ from "stacktsk.h" to this file.

None

3. Added PORTA defs.

None

4. htnicee.pjt 1. Removed "ee256.c".

None

2. Added "udp.c", "dhcp.c", "ftp.c", "xlcd.c", "xeprom.c" files

Add these files if needed.

5. htnicpg.pjt None

6. htslee.pjt 1. Removed "ee256.c".

None

2. Added "ftp.c", "xlcd.c", "xeprom.c" files

None

7. http.c 1. Included compile-time verification that HTTP module is included.

None

2. Put HTTP message strings into one array "HTTPMessages".

None

3. Modified to return "Service Unavailable" message if MPFS is being

None

remotely programmed.

4. Modified SendFile() to make use of sequential EEPROM read.

None

8. ip.c 1. Added one more paramter to IPGetHeader() to support IP Gleaning
Custom apps using IP needs to be

modified.

9. mac.c 1. Replaced fixed delay routines with CLOCK_FREQ dependent

None

routines

10. mpfs.c 1. Replaced ee256.h with xeeeprom.h.

None

2. Added MPFSFormat(), MPFSPut() etc. routines

None

3. Added sequential read and page write operations

Custom apps using MPFS directly

needs to be modified.

4. Defined MPFS_WRITE_PAGE_SIZE for MPFSPut operations.

Apps using different EEPROM page size

needs to be modified.

11. mpnicee.pjt 1. Removed "ee256.c"

None

2. Added "xcld.c", "xeeeprom.c" files

None

12. stacktsk.c 1. Replaced ee256.h with xeeeprom.h

None

2. Added IP Gleaning and DHCP support.

None

13. stacktsk.h 1. Moved CLOCK_FREQ to compiler.h

None

2. Added STACK_USE_DHCP, STACK_USE_FTP_SERVER etc. options

None

3. Added compile-time enable/disable of modules based on selection of higher level modules.

None

4. Modified MY_DEFAULT_MAC_BYTE? to use Microchip OUI id.

None

5. Added compiler-time check to confirm available TCP sockets

None

6. Added MSB and LSB macros.

None

7. Added SerialNumber etc. to AppConfig structure

None

8. Commented module selection defines: They are defined by compiler

None

command-line options.

Real application should define them here in this file.

14. tcp.c 1. Moved TCP_STATE and TCP_INFO to .h file.

None

2. Fixed TCPIsConnected()

None

3. Fixed TCPDisconnect()

None

4. Modified TransmitTCP() to set receive window of one segment

None

5. Modified TransmitTCP() to use max segment size equal to predefined value.

None

6. Improved TCP State machine

None

15. tick.c 1. Modified TICK type to 16-bit.

None

2. Made use of TICK_PRESCALE_VALUE

None

3. Added code to blink PICDEM.net "System LED"

Remove if not required.

16. webservr.c 1. Added LCD support

N/A

2. Made TickUpdate() on Timer0 interrupt

N/A

3. Added code to save/restore board configuration

N/A

4. Added board setup via RS-232.

N/A

5. Added call to FTP modules

If needed, add this.

1.4 Utilities

Describes PC software applications included with the TCP/IP Stack.

Description

This section will discuss the computer software applications included with Microchip's TCP/IP Stack.

These tools are implemented using the C# or Java programming languages, or both. The C# tools (*.exe) will require the Microsoft® .NET Framework v2.0 to be installed on the local PC. The Java tools (*.jar) require Java Runtime Environment (JRE) 1.6 or later to be installed on the target computer.

1.4.1 MPFS2

Documentation for the MPFS2 utility

Description

The MPFS2 Utility packages web pages into a format for efficient storage in an embedded system. It is a graphical application for PCs that can generate MPFS2 images for storage in external storage or internal Flash program memory.

When used to build MPFS2 images, the MPFS2 Utility also indexes the dynamic variables found. It uses this information to generate `http_print.h`, which ensures that the proper callback functions are invoked as necessary. It also stores this index information along with the file in the MPFS2 image, which alleviates the task of searching from the embedded device.

Finally, when developing an application that uses external storage, the MPFS2 Utility can upload images to the external storage device using the upload functionality built into the HTTP2 web server or FTP server.

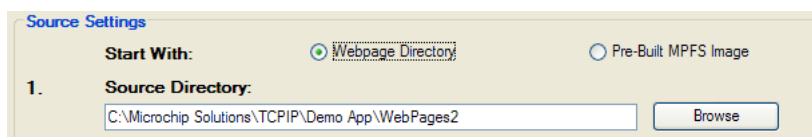
The source code for this application is included in the Microchip Applications Libraries installer.

1.4.1.1 Building MPFS2 Images

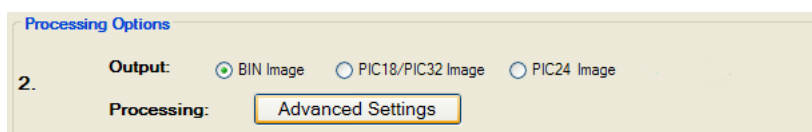
Describes how to build an MPFS Image

Description

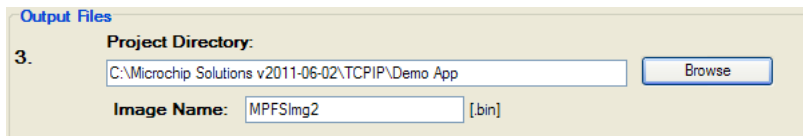
The MPFS2 Utility has four steps, which are denoted on the left hand side of the dialog. To build an MPFS image, select **Start With: Webpage Directory** in step 1 and choose the directory in which the web pages are stored.



Step 2 selects the output format. If storing the web pages in external EEPROM or serial Flash, choose the **BIN Image** output format. If internal program memory will be used, select **PIC18/PIC32 Image** for use with 8-bit and 32-bit parts, or **PIC24 Image** for 16-bit targets.

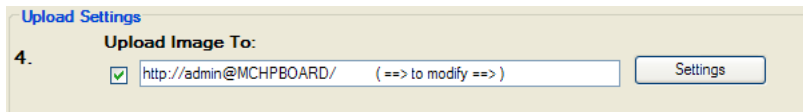


Step 3 asks for the MPLABX IDE project directory. The MPFS tool will write the image file to the project directory, and will also update the `http_print.h` file there if needed. Select the correct directory so that the right files are modified.



Step 4 controls the upload settings. When external EEPROM or serial flash is used for storage, the option to upload the newly created image to the board is available. Check the box next to **Upload Image To** to enable this feature. The target host name (or IP address), upload protocol, and upload path may need to be changed to the one chosen when the board was first configured. You may also need to modify the user name and password used to access the secured functionality in your application, like web page upload. Use the **Settings** button to edit these values.

If internal program memory is being used, the image will be compiled in with the project and so direct uploads are not available. Make sure to include the output source file indicated in step 3 as part of the project.



Once all the correct settings have been chosen, click the **Generate** button to create the image. If uploads are enabled, this will also attempt to upload the file to the device.

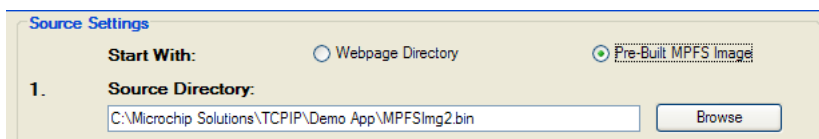
1.4.1.2 Uploading Pre-built MPFS2 Images

Procedure to upload images to external storage

Description

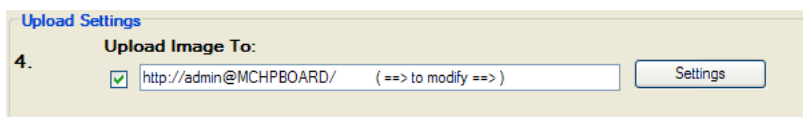
There are two ways to upload a pre-built image to external storage. The first is described in the Getting Started section, and involves uploading from the browser directly. The second is to use the MPFS2 Utility to upload the image. You can select HTTP or FTP uploading to match the protocol that your application uses.

To use the MPFS2 Utility to upload an image, begin by selecting **Start With: Pre-Build MPFS Image** in step 1 at the top. Choose the image file to upload.



Steps 2 and 3 are not required for pre-built images. Proceed directly to step 4 and verify that the upload settings are correct. The target host name (or IP address), upload protocol, and upload path may need to be changed to the one chosen when the board was first configured. You may also need to modify the user name and password used to access the secured functionality in your application, like web page upload. Use the **Settings** button to edit these values.

Once all the settings are correct, click the **Upload** button. The image will be uploaded to the board.

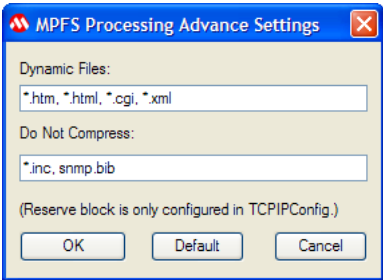


1.4.1.3 Advanced MPFS2 Settings

Working with advanced configuration options

Description

The **Advanced Settings** dialog found in step 2 provides greater control over how files are processed.



The **Dynamic Files** list indicates which file types to parse for dynamic variables. By default, all files with the extensions htm, html, cgi, or xml are parsed. If an application has dynamic variables in other file types, these types must be added to the list. This field must be a comma-separated list of extensions and file names.

The **Do Not Compress** field indicates which file types should never be compressed. Compressing files with GZIP saves both storage space and transmission time. However, this is only suitable for static content such as CSS or JavaScript. Any files with dynamic variables will automatically be excluded. In addition, any file that the PIC may need to process internally should be excluded. Files included via ~inc:filename~ should not be compressed, nor should any BIB file used for the SNMP module (if present). Additional file types can be added to this list if a custom application will be accessing the MPFS.

The GZIP compressor will attempt to shrink all files. In some cases, especially with images, little or no compression is achieved. When this occurs the file is stored as-is in the MPFS image.

1.4.1.4 MPFS2 Command Line Options

How to use the MPFS2 Utility from the command line

Description

To facilitate batch files and automation, the MPFS2 Utility also supports execution from the command line. The syntax is as follows:

```
MPFS2.jar [options] <SourceDir> <ProjectDir> <OutputFile>
```

The **SourceDir**, **ProjectDir**, and **OutputFile** options are required and should be enclosed in quotation marks. The **OutputFile** option will be relative to **ProjectDir**, and **cannot** be a full path name.

The various option switches are described in the table below:

Switch	Short	Description
/BIN	/b	Output a BIN image (Default)
/C18_C32	/c	Output a PIC18 or PIC32 image
/ASM30	/s	Output an PIC24 image
/mpfs2	/2	Use the MPFS2 format (Default)
/html "..."	/h "..."	File types to be parsed for dynamic variables (Default: "*.htm, *.html, *.cgi, *.xml")
/xgzip "..."	/z "..."	File types to be excluded from GZIP compression (Default: "*.bib, *.inc")

The command-line interface does not support image uploads. For batch or production uploads, use a tool such as `wget` to upload the generated BIN image.

1.4.2 Microchip TCP/IP Discoverer

Describes the Microchip TCP/IP Discoverer project.

Description

The Microchip TCP/IP Discoverer PC project (formerly known as the Embedded Ethernet Device Discoverer) will aid in embedded product device discovery (with the Announce protocol) and will demonstrate how to write PC applications to communicate to embedded devices.

When the "Discover Devices" button is clicked, this application will transmit a broadcast UDP packet containing the message, "Discovery: Who is out there?" on the local network to port 30303. If any embedded devices with the Announce protocol enabled are connected to the network, they will respond with a UDP packet containing their host name (NBNS) and MAC address.

The Java source code for this application is also included. This source code should provide a rough idea of how to write a PC-based application to communicate with your embedded devices.

1.5 Getting Started

Describes the steps necessary to begin using the TCP/IP Demo Applications.

Description

This section describes the steps necessary to begin using Microchip's TCP/IP Demo Applications. This section contains specific information for setting up and using the generic TCPIP Demo App. Most of this setup information can be applied to get started with other demo applications as well.

1.5.1 Peripheral Usage

Summary

Describes the peripherals required by the TCP/IP stack.

Description

Several microcontroller peripherals can/must be used to implement a TCP/IP stack application.

Type	Specific/Configurable	Polled/Interrupt	Purpose
Timer	Timer 0 for PIC18, Timer 1 otherwise	Interrupt	Used to implement a tick timer
SPI or PMP	Select via #define in system_config.h and system.h. See Hardware Configuration.	Polled	The SPI or PMP module is used to drive network transceivers.
SPI	Select via #define in system_config.h and system.h. See External Storage.	Polled	Used to interface to an EEPROM or Serial Flash chip, as an option to store web pages for MPFS2 or the AppConfig structure.
SPI	Select via #define in system_config.h and system.h. See External Storage.	Polled	Used to interface to a serial RAM as a optional socket allocation method.

1.5.2 Demo Board Information

This section gives a brief introduction and links to more information for the Wi-Fi demo boards.

Description

1.5.2.1 PIC24FJ256GB110 Plug-In-Module (PIM)

Summary

Processor module for the PIC24 family.

Description

PIC24FJ256GB110 Plug-In-Module (PIM)

Overview

The PIC24FJ256GB110 PIM is designed to demonstrate the capabilities of the PIC24FJ256GB110 family using the Explorer 16 Demonstration Board kit and the PICtail Plus Daughter Boards. The PIC24FJ256GB110 is specifically designed to be used in conjunction with the USB PICtail Plus Daughter Board to allow development of USB applications. Because a few pins on the device are dedicated to the USB module, several of the existing features of the Explorer 16 must be rerouted on the PIC24FJ256GB110 PIM. The Peripheral Pin Select (PPS) feature, available on this device, allows the existing peripherals to be remapped to the new I/O pins.

More Information

[Product Information](#)

1.5.2.2 PIC18F87J11 Plug-In-Module (PIM)

Summary

Processor module for the PIC18 family.

Description

PIC18F87J11 Plug-In-Module (PIM)

Overview

Ideal for 3V applications that are cost sensitive, requiring flexible serial communication with four serial ports: double synchronous serial ports (I²C and SPI) and double asynchronous serial ports. Large amounts of Flash program memory make it ideal for instrumentation panels, TCP/IP enabled embedded applications as well as metering, industrial control and monitoring applications. The PIC18F87J11 is an enhancement to the PIC18F87J10 family, including an internal 8MHz oscillator, lower power and improved EEPROM emulation via word Flash write capabilities. Features

48 MHz and 12 MIPS at 3V Flash 10,000 endurance, 20 years retention 2.0 - 3.6V voltage range 5V tolerant digital inputs Enhanced ICD with 3 HW breakpoints Internal 8MHz oscillator nanoWatt technology Parallel Master Port (PMP)

More Information

[Product Information](#)

1.5.2.3 PIC32MX795F512L Plug-In-Module (PIM)

Summary

Processor module for the PIC32 family.

Description

PIC32MX795F512L Plug-In-Module (PIM)

Overview

MCU Core 80MHz/105DMIPS, 32-bit MIPS M4K® Core USB 2.0 On-The-Go Peripheral with integrated PHY 10/100 Ethernet MAC with MII/RMII Interfaces 2 x CAN2.0b modules with 1024 buffers 8 Dedicated DMA Channels for USB OTG, Ethernet, and CAN 5 Stage pipeline, Harvard architecture MIPS16e mode for up to 40% smaller code size Single cycle multiply and hardware divide unit 32 x 32-bit Core Registers 32 x 32-bit Shadow Registers Fast context switch and interrupt response MCU System Features 512K Flash (plus 12K boot Flash) 128K RAM (can execute from RAM) 8 Channel General Hardware DMA Controller Flash prefetch module with 256 Byte cache Lock instructions or data in cache for fast access Programmable vector interrupt controller Analog Features Fast and Accurate 16 channel 10-bit ADC, Max 1 Mega sample per second at +/- 1LSB, conversion available during SLEEP & IDLE Power Management Modes RUN, IDLE, and SLEEP modes Multiple switchable clock modes for each power mode, enables optimum power settings Debug Features iFlow Trace: Non-intrusive Hardware Instruction Trace port (5 Wires) 8 hardware breakpoints (6 Instruction and 2 Data) 2 wire programming and debugging interface JTAG interface supporting Programming, Debugging and Boundary scan Other MCU Features Fail-Safe Clock Monitor - allows safe shutdown if clock fails 2 Internal oscillators (8MHz & 31KHz) Hardware RTCC (Real-Time Clock and Calendar with Alarms) Watchdog Timer with separate RC oscillator Pin compatible with 16-bit PIC® MCUs Serial Communication Modules allow flexible UART/SPI/I2C™ configuration

More Information

[Product Information](#)

1.5.2.4 MRF24WB/MRF24WG PICtail Daughter Board

Summary

MRF24WB PICTail daughter board

Description/

The MRF24W Wi-Fi PICTail Plus Daughter Board is a demonstration board for evaluating Wi-Fi connectivity on boards with a PICTail or a PICTail Plus connector. The board features the Microchip MRF24WB0MA (802.11b: 1 to 2Mbps) or MRF24WG0MA (802.11b/g) module, which includes a Wi-Fi transceiver and associated circuit elements.

MRF24WB0MA supports both infrastructure and adhoc network types. MRF24WG0MA supports more extensive features covering infrastructure, adhoc, Wi-Fi Direct (peer-to-peer) and softAP network types. In addition, MRF24WG0MA supports Wi-Fi Protected Setup (WPS).

Visit the Microchip Web Site to view more information on www.microchip.com/WiFi , www.microchip.com/AC164149 and www.microchip.com/AC164136-4

These products are compatible with the Explorer16 Development Board and PIC18 Explorer Development Board.

More Information

[Product Information](#)

1.5.3 Wi-Fi Demo Board Hardware Setup

Walks through hardware configuration for supported development platforms

Description

The first step to use the stack is to make sure an appropriate development board is configured. To get started, select a platform from the topics presented below.

1.5.3.1 Explorer 16

Summary

Development platforms for 16- and 32-bit parts using external MRF24WB0M / MRF24WG0M MAC+PHY.

Description

The Explorer 16 board is an all-purpose demonstration and development board for 16-bit and 32-bit parts. It can be expanded for TCP/IP support using the MRF24WB or MRF24WG Wi-Fi PICtail Plus daughter board.

WiFi demo application is configured to run on Explorer 16 with PIC24FL256GP110 or PIC32MX795F512L.

Before using the Explorer 16, check that:

1. Switch S2 selects PIM
2. Jumper J7 selects PIC24 (even though the label reads PIC24, this jumper setting selects the programming signals to any PIC on the Explorer 16).
3. Plug-in the PIC24FL256GP110 or PIC32MX795F512L PIM

Using the Microchip MRF24WB0MA / MRF24WG0MA Wi-Fi PICtail

The Explorer 16 can be used to debug wireless functionality by connecting the PICtail as show in the pictures, with header J2 on the PICtail inserted into the top slot of connector J5 (Explorer 16).

Note if jumper JP3 exists, it must be shorted between pins 1 and 2 when used on this development platform.

Once your hardware is configured, you can program your board with your preferred demo project. The next few topics in the Getting Started section of this help file provide a tutorial for setting up the generic TCP/IP demo application.

More Information

Explorer 16 www.microchip.com/explorer16.

1.5.3.2 PIC18 Explorer Development Board

Summary

Development platform for 8-bit parts using external MRF24WB0MA / MRF24WG0MA MAC+PHY.

Description

The PICDEM.net 2 is the recommended development board for this part.

When using the PIC18 Explorer, ensure that jumpers JP2 and JP3 are shorted to enable the LCD and EEPROM, and switch S4 is configured to properly select the on-board PIC or the ICE setting, as your application requires.

Note if jumper JP3 exists, it must be shorted between pins 2 and 3 when used on this development platform.

Once your hardware is configured, you can program your board with your preferred demo project. The next few topics in the Getting Started section of this help file provide a tutorial for setting up the generic TCPIP demo application.

More Information

PIC18 Explorer Part Number: DM183032.

www.microchip.com/pic18explorer.

1.5.3.3 Wi-Fi G Demo Board

Summary

Describes the wireless 802.11b/g demo board based on Microchip 802.11b/g MRF24WG0MA RF transceiver and PIC32MX695F512H processor to provide Wi-Fi connectivity.

Description

Wi-Fi G Demo Board provides a low-cost and portable development system for Microchip MRF24WG0MA 802.11b/g RF Transceiver. The Wi-Fi G Demo Board is preloaded with the demo software for the user to explore the features of the MRF24WG0MA RF Transceiver. It is also expandable through a 8-pin expansion port interface, which allows the user to extend its functionality by adding various sensor expansion circuit designs.

The Wi-Fi G Demo Board is a compact demonstration platform for customers to easily evaluate and configure Microchip's new MRF24WG0MA Wi-Fi module. The demo board is a fully-functional standalone web server powered by 2 AAA batteries. It comes with a PIC32 pre-programmed with the Microchip TCP/IP stack, connected to an onboard, fully-certified MRF24WG0MA Wi-Fi module.

More Information

[Product Information](#)

1.5.4 Programming and First Run

Programming the device and clearing EEPROM for first run

Description

Once the hardware is configured, you are ready to program the device for the first time.

Project Setup

Use MPLAB X IDE to program the hex file in the board.

Note that the projects and source code used to build each hex file are present in the <mla_installation_root>\apps\tcpip directory. The hardware and firmware configuration files used to build each project are present in the demo directory.

Programming

Select your device programmer from the Programmer menu in MPLABX IDE, and then use the Program shortcut button or

the Program menu option to program the code you imported to your board.

Clearing the EEPROM

The TCP/IP Stack stores network configuration settings (such as the host name, MAC address, default static IP addresses, SNMP strings, WiFi network name (SSID), etc) in external EEPROM on the board. The demo project will detect if the default values have been changed in the EEPROM, and if so, use the new values. If not, the demo will use the default values configured in `tcpip_config.h` and `drv_wifi_config.h`. Checksums stored in the EEPROM are used to determine if the structures stored in EEPROM are valid. Manually clearing the EEPROM will allow the demo to resume using the default settings.

Use the following procedure to clear the EEPROM:

1. Make sure the development board is programmed and not in debug mode
2. Disconnect the programmer/debugger from the board
3. Press and hold BUTTON0 (RD13/S4 on Explorer 16 or RB3/S5 on Explorer 16™ 2)
4. Press and release the MCLR button
5. Continue holding BUTTON0 until several LEDs flash indicating that EEPROM has been cleared. This takes about 4 seconds.
6. Release BUTTON0
7. Press and release MCLR again to reset the software

Once you see LED0 (right-most LED) blinking, the software is running and ready for use.

If you are using the MRF24WB0M / MRF24WG0M WiFi PICtail, you'll need to configure your wireless access point first. For all Ethernet devices, Connect your Development Board to your network.

1.5.5 Configure your WiFi Access Point

Configure the access point to work with the development kit. For information on configuring the MRF24WB0M / MRF24WG0M for your AP refer to "Configuring WiFi Security"

Description

To run the Wi-Fi demos with the MRF24WB0M / MRF24WG0M PICtail, you'll also need to setup a wireless access point. As an example, this guide will walk through the setup of a Linksys WRT54G2 access point.

Access Point Browser GUI

The Linksys, along with many other popular router brands, uses a built-in webserver on the router to administer the network (both wired and wireless). Please consult the documentation that came with your router for more information on configuration and setup. For a list of known compatible routers refer to section "Access Point Compatibility". To gain access to this web page, you'll need to point your browser to `http://192.168.1.1`. By default, the username field is left blank, and the password is admin.

Wireless Setup

Along the top of the webpage, there should be many tabs for all the different features of the access point. One of the tabs should read "Wireless". After clicking the tab, you will be presented with the Wi-Fi protected setup page. You'll need to click the manual tab to be able to enter your own wireless settings to match the demo.

The out of box demo is looking for an AP with the following parameters (note that the SSID is case sensitive):

SSID	MicrochipDemoAP
Security	None
Channel	1,2,3,4,5,6,7,8,9,10,11

You should have settings similar to the following:

Once the network is setup, you can connect your device to the network.

1.5.6 Connecting to the Network

Connecting your development board to the network

Description

All devices on a TCP/IP network must be assigned an IP address. Whereas the MAC address is the hardware address of the device, the IP address is a software address. The DHCP (Dynamic Host Configuration Protocol) allows this assignment to take place automatically (for more address information and configuration options, see the Addresses topic).

The demo application comes with both a DHCP server and DHCP client configured. This allows the board to connect to most networks without configuration. If a free Ethernet port is available on a nearby router, switch, or wall plate, the board can be connected directly using any standard straight-through Ethernet cable. Under this configuration, the board will attempt to obtain an IP address from your network's DHCP server.

If this method is not possible, a crossover Ethernet cable can be used to connect the board directly to a PC's Ethernet port. Using this configuration, the board will act as its own DHCP server and will assign a single IP address to the computer. (The Fast 100Mbps Ethernet PICtail Plus and some newer PCs do not require a special crossover cable, so any Ethernet cable can be used.)

Connect the development board to the network and wait for the link LED on the Ethernet jack to light up. The board is now on the network and capable of communicating with other devices.

If the link LED on the Ethernet jack does not light, your board cannot link to the network. Ensure that you have selected the proper cable, and try switching from a straight-through to a crossover cable, or vice versa.

Now that the board is online, you can Upload the Demo Web Pages.

1.5.7 Uploading Web Pages

Uploading web pages to the device

Description

Web pages are stored as an MPFS2 image. This image can be placed in either external non-volatile storage (EEPROM or SPI Flash), or in the microcontroller's internal Flash program memory. For this example, the EEPROM chip ([25LC256](#)) on your demo board will be programmed with a pre-built MPFS2 BIN image. This location can be changed via a compile-time option in `tcpip_config.h`.

The target application on the development board must be running for this procedure to work. Make sure the right most status LED is blinking.

Each hex file is configured to provide a **Host Name** for your development board. This will be the name by which your board is accessed. In the default hex files, the host name is `mchpboard`, so your board can be accessed at `http://mchpboard`. This host name uses the NetBIOS Name Service. It is only available on your local subnet, and will not be accessible from the Internet. Note that this service is not supported by all operating systems. If you have difficulty accessing your board, try using the IP address shown on the LCD screen instead (e.g. access the board at `http://192.168.1.101`). You can also determine the IP address by using the Microchip TCP/IP Discoverer.

Open a web browser and access the board at `http://mchpboard/mpfsupload`. This form will allow web pages stored on the device to be updated. If you mistype this URL, the board will provide a default HTTP 404 error page with a link to the MPFS Upload page. This default 404 page will not appear if you've configured your browser to override custom error pages (e.g. by checking "Show friendly HTTP error messages" in Internet Explorer 7's internet options menu). Select the file

MPFSImg2.bin from the TCPIP\Demo App folder as shown below.

This update method is only available when using external storage.

When the **Upload** button is clicked, the MPFS image is sent to the board and programmed into the EEPROM. As this happens, the activity LED on the Ethernet jack will blink. Once the browser reports that the upload has completed, click the link provided within the status message to access the board's web pages.

You can now Access the Demo Application.

1.5.8 Accessing the Demo Application

Accessing the demo application

Description

The board is now accessible at the `mchpboard` host name or at the board's IP address. When accessed in a web browser, a real-time update of the board's controls is displayed. The demo application will show off several features, and will explain how to modify the web pages and application to suit various needs.

If you attempt to access the Network Configuration or SNMP Configuration web pages from the red menu on the left, you will be prompted for a username and password. The default username is "admin" and the default password is "microchip". More information is available on the Authentication web page, or in the HTTP2 server authentication help topic.

Some features of the default demo application may not be available on certain hardware platforms. For more information, see the TCPIP Demo App features by Hardware Platform topic. For information about how to use each feature of the TCP/IP Demo Application, consult the subtopics in the TCP/IP Demo Application Demo Modules topic.

Once you have finished exploring the demo application, you can proceed to the Stack API section to learn more about the stack and start developing your own application.

If you are exploring the TCPIP WiFi demo applications and want to set up security, you can get more information on the WLAN security page.

1.5.9 Configuring WiFi Security

Configures the wireless access point and demo code for wireless security

Description

The MRF24WB0M / MRF24WG0M can be configured to connect to wireless networks with encryption enabled. Both MRF24WB0M / MRF24WG0M supports WEP (40-bit and 104-bit), as well as WPA-PSK (TKIP) and WPA2-PSK (TKIP/AES). In addition, MRF24WG0M supports Wi-Fi Protected Setup (WPS) both Push Button Configuration (WPS-PBC) and Personal Information Number (WPS-PIN).

Device Security Modes

Security settings for the MRF24WB0M / MRF24WG0M are located in the file `drv_wifi_config.h`. To enable security features the `#define` preprocessor definition for `MY_DEFAULT_WIFI_SECURITY_MODE` must be defined as one of the following options:

WF_SECURITY_WEP_40	40-bit WEP security. This equates to 5 ASCII characters or 10 hex digits. MY_DEFAULT_WEP_KEYS_40 only supports one key that can be programmed (default is key 0).
WF_SECURITY_WEP_104	104-bit WEP security. This equates to 13 ASCII characters or 26 hex digits. MY_DEFAULT_WEP_KEYS_104 only supports one key that can be programmed (default is key 0).
WF_SECURITY_WPA_AUTO_WITH_KEY	Uses the 32 bytes in MY_DEFAULT_PSK as the key to join the network. These values are generated from a hash of the SSID name and WPA passphrase. For the purpose of the demo, the 32-bytes in MY_DEFAULT_PSK in drv_wifi_config.h correspond to an SSID of "MicrochipDemoAP" and passphrase "Microchip 802.11 Secret PSK Password".
WF_SECURITY_WPA_AUTO_WITH_PASS_PHRASE	Instructs the MRF24WB0M / MRF24WG0M to generate the 32 byte PSK using the SSID and passphrase. The default in drv_wifi_config.h corresponds to an SSID of "MicrochipDemoAP" and passphrase "Microchip 802.11 Secret PSK Password". Note that it takes approximately 30 seconds for the MRF24WB0M and 25 seconds for the MRF24WG0M to calculate this value ^[1] .
WF_SECURITY_WPS_PUSH_BUTTON WF_SECURITY_WPS_PIN	Supported by MRF24WG0M only. For WPS-PBC, define the MY_DEFAULT_SSID_NAME as "". For WPS-PIN, define the MY_DEFAULT_WPS_PIN to be the same as the AP/router PIN, for example, 12390212 and define the MY_DEFAULT_SSID_NAME to be the same as the AP or router's SSID.

Note: Some routers try to increase the random nature of the WEP key by adding an additional layer that will convert an ASCII passphrase into a hexadecimal key. The MRF24WB0M / MRF24WG0M PICtail will require a hexadecimal key, no matter which way it is generated.

Access Point Security Settings

The access point will also need to be changed to match the same security settings. Wireless security settings can be found in the "Wireless Security" tab under the main "Wireless" tab (example shows a Linksys WRT5G2). The drop-down box for security has all the different security options. Note that for WPA/WPA2, the MRF24WB0M / MRF24WG0M only supports personal security levels (as opposed to enterprise, which is not supported).

[1]: Once the 32-byte PSK is calculated, it can be retrieved by the host from the MRF24WB0M / MRF24WG0M. The host can then save this key to external non-volatile memory. On future connection attempts, the host can program the MRF24WB0M / MRF24WG0M with the WF_SECURITY_WPA/WPA2/WPA_AUTO_WITH_KEY options, provide the saved key, and not have to wait 30 seconds to reconnect to the network.

Pre-generated PSK

You also have the option to pre-generate the PSK and use the 32-byte PSK directly in the source code. One handy tool to generate the PSK can be found online at the Wireshark Foundation <http://www.wireshark.org/tools/wpa-psk.html>. The Wireshark website can generate the expected 32-byte PSK key with the SSID name and the passphrase. You can then use these values in the variable MY_DEFAULT_PSK in tcpip_config.h.

Wi-Fi Protected Setup (WPS)

WiFi Protected Setup (WPS) allows users to set up and expand the WiFi networks with security enabled, even if they are not familiar with the underlying technologies or processes involved. For example, users no longer have to know that SSID refers to the network name or WPA2 refers to the security mechanism. WPS will configure the network name SSID and security key for the AP and WPS client devices on a network. It supports the WEP, WPA-PSK, WPA2-PSK, and WPA-PSK/WPA2-PSK Auto/Mixed security methods.

AP/routers from 2007 onwards will have this WPS feature.

1.6 Demo Information

Describes the stack demos.

Description

This section describes Microchip's TCP/IP Demo projects, including information about demo-hardware compatibility. For information about how to load and configure the demos, please consult the Getting Started section.

1.6.1 Demo Compatibility Table

Describes the hardware platforms that are compatible with each Demo.

Description

Each stack demonstration project comes with several predefined, tested configurations. Pre-built hex files for each demo are available in the **hex** subdirectory in that demo's project folder. This section will specify the combinations of demo boards, processors, MAC/PHY layers, and communication buses that are set up to work by default.

TCPIP WiFi Demo App

Demo Board	Processor	MAC/PHY Layer	Comm Bus	Notes
PIC18 Explorer	18F87J11	MRF24WB0M / MRF24WG0M	SPI	
Explorer 16	24FJ256GB110	MRF24WB0M / MRF24WG0M	SPI	
Explorer 16	32MX795F512L	MRF24WB0M / MRF24WG0M	SPI	

TCPIP WiFi G Demo App

Demo Board	Processor	MAC/PHY Layer	Comm Bus	Notes
Wi-Fi G	32MX695F512H	MRF24WG0M	SPI2	

TCPIP WiFi EasyConfig Demo App

Demo Board	Processor	MAC/PHY Layer	Comm Bus	Notes
Explorer 16	24FJ256GB110	MRF24WB0M / MRF24WG0M	SPI	
Explorer 16	32MX795F512L	MRF24WB0M / MRF24WG0M	SPI	

TCPIP WiFi Console Demo App

Demo Board	Processor	MAC/PHY Layer	Comm Bus	Notes
Explorer 16	24FJ256GB110	MRF24WB0M / MRF24WG0M	SPI	
Explorer 16	32MX795F512L	MRF24WB0M / MRF24WG0M	SPI	

1.6.2 Available Demos

The TCP/IP Stack comes with several example applications. These applications are described in the following sections.

1.6.2.1 TCPIP WiFi G Demo

Implements a wireless 802.11b/g low-cost and portable application to demonstrate Wi-Fi capabilities using MRF24WG0MA and PIC32MX695F512H processor.

Description

The Wi-Fi G Demo Board utilizes the EasyConfig mechanism in TCPIP WiFi EasyConfig Demo App, which allows configuration of an embedded device on a wireless network. It utilizes the web server of the TCP/IP stack to allow the user to input the desired network information from a client browser, and then reset the device to connect to the desired network. By default, Wi-Fi G Demo Board will start up in SoftAP network mode.

Refer to the documentation [Wi-Fi G Demo Board User's Guide](#) for more information.

1.6.2.2 TCPIP WiFi Console Demo App

Implements a wireless console demo application that accepts client commands.

Description

The TCPIP WiFi Console Demo App (previously the TCPIP WiFi Iperf Demo App) is a command line interface (CLI) to the MRF24WB0M / MRF24WG0M. It allows for command line debugging and setup of network information for the wireless LAN. It also has iperf built in for doing WLAN bandwidth testing. This application is meant more as a development debug tool, and should be disabled in end user applications.

The following network types are supported

- CFG_WF_INFRASTRUCTURE (as a client in infrastructure network)
- CFG_WF_ADHOC
- CFG_WF_P2P (Wi-Fi Direct) (as a group client in Wi-Fi Direct network)

CFG_WF_P2P is only available for MRF24WG0MA/B.

New security modes are supported.

- WF_SECURITY_WPS_PUSH_BUTTON (supported by MRF24WG0M only)
- WF_SECURITY_WPS_PIN (supported by MRF24WG0M only)

Wireless configurations are set up in `drv_wifi_config.h`

1.6.2.2.1 Standalone Commands

Standalone CLI commands for talking with the MRF24WB0M / MRF24WG0M.

Description

These command line interface (CLI) commands are not related to the wireless or networking interface directly.

<code>help</code>	Lists all the available CLI commands for the MRF24WB0M / MRF24WG0M.
<code>getwfvver</code>	Lists the MRF24W firmware version and host driver version numbers.
<code>reset</code>	Issues a host reset.
<code>cls</code>	Resets the prompt.
<code>iperf</code>	Initiates an iperf session. See the section on iperf for more information.
<code>kill iperf</code>	Stops a running iperf session. See the section on iperf for more information.
<code>ping</code>	Initiates a ping session. This will verify IP-level connectivity to another TCP/IP terminal by sending Internet Control Message Protocol (ICMP) Echo Request messages. The receipt of corresponding Echo Reply messages are displayed, along with round-trip times.
<code>killping</code>	Stops a running ping session.
<code>wpscred</code>	Display WPS credentials.

1.6.2.2.1.1 iwconfig Commands

Commands for controlling the wireless network interface.

Description

Note that most of these items should not be changed while the device is in a connected state to a network.

iwconfig commands take the following structure:


```

iwconfig [ ssid <name> ]
         [ mode <idle|managed|adhoc> ]
         [ channel <channel list|all> ]
         [ power <reenable|disable|unicast|all> ]
         [ domain <name> ]
         [ rts <length> ]
         [ scan ]
         [ hibernate ]
         [ wakeup ]

```

Note: iwconfig with no options will display wireless status.

ssid	
name	1-32 ASCII characters. Currently doesn't accept spaces in the SSID name.

mode	
idle	Forces the MRF24WB0M / MRF24WG0M to disconnect from any currently connected network (adhoc or infrastructure).
managed	The MRF24WB0M / MRF24WG0M will connect to the SSID in infrastructure mode. Note that all the network parameters must be correct before this command is called.
adhoc	The MRF24WB0M / MRF24WG0M will connect to the SSID in adhoc mode. Note that all the network parameters must be correct before this command is called.

channel	
channel list	A comma separated list of all the channels to scan.
all	Sets the MRF24WB0M / MRF24WG0M to scan all channels in the given regulatory domain.

power	
reenable / all	Enables all power saving features (PS_POLL) of the MRF24WB0M / MRF24WG0M. MRF24WB0M / MRF24WG0M will wake up to check for all types of traffic (unicast, multicast, and broadcast).
disable	Disables any power savings features. The MRF24WB0M / MRF24WG0M will always be in an active power state.
unicast	The MRF24WB0M / MRF24WG0M will be in it's deepest sleep state, only waking up at periodic intervals to check for unicast data. The MRF24WB0M / MRF24WG0M will not wake up on the DTIM period for broadcast or multicast traffic.

domain	
fcc	United States channels 1-11.
ic	Canada channels 1-11. Applicable for MRF24WB0M only.
etsi	European channels 1-13.
spain	Spanish channels 10-11. Applicable for MRF24WB0M only.
france	French channels 10-13. Applicable for MRF24WB0M only.
japan	Japanese channel 1-14. Applicable for MRF24WG0M only.
japana	Japanese channel 14. Applicable for MRF24WB0M only.
japanb	Japanese channels 1-11. Applicable for MRF24WB0M only.

rts	
length	Set the requested number of bytes to send. Default max is 2347.
scan	
	Instructs the MRF24WB0M / MRF24WG0M to perform an active site scan. Scan results will be displayed to the output terminal.
hibernate	
	Turns off LDO of the MRF24W module, which is equivalent to removing power to the MRF24WB0M / MRF24WG0M. Has the same effect of resetting MRF24WB0M / MRF24WG0M. MRF24W state is not maintained when transitioning to hibernate mode.
wakeup	
	Restores power to the MRF24WB0M / MRF24WG0M and reconnects.

Note: scan is only supported by the WiFi EasyConfig demo.

1.6.2.2.1.2 ifconfig Commands

Commands for controlling the network interface.

Description

Note that these items should not be changed while the device is in a connected state to a network.

ifconfig commands take the following structure:

```
ifconfig [ <IP address> ]
        [ <MAC address> ]
        [ netmask <IP address> ]
        [ gateway <IP address> ]
        [ auto-dhcp <start|drop> ]
```

Note ifconfig by itself will give network status.

IP address	
	Use a static IP address. IP address must be in dot-decimal notation. Note that this command will return an invalid parameter if the DHCP client is enabled. First disable the DHCP attempts (ifconfig auto-dhcp drop) before running this command.
MAC address	
	Redefine the device MAC address. MAC address must be specified in hexadecimal colon notation. This command can only be issued when the MRF24WB0M / MRF24WG0M is in idle mode. Doing so at other times can have unexpected results.
netmask	
IP address	Use the specified IP address for the netmask. The netmask value is specified in dot-decimal notation.

gateway	
IP address	Configure the gateway address. The gateway value is specified in dot-decimal notation.

auto-dhcp	
start	Starts the DHCP client. Only valid if the DHCP module has been compiled in. DHCP client is started by default.
drop	Stops the DHCP client. A static IP address will need to be assigned to the device. Only valid if the DHCP module has been compiled in.

1.6.2.2.1.3 iwpriv Commands

Commands for controlling the wireless encryption settings.

Description

Note that these items should not be changed while the device is in a connected state to a network.

iwpriv commands take the following structure:

```
iwpriv [ enc <none|wep|wpa-psk|wpa-phrase> ]
      [ key <[1][2][3][4]> <value> ]
      [ psk <value> ]
      [ phrase <value> ]
```

Note iwpriv by itself will display network security settings.

enc	
none	The MRF24WB0M / MRF24WG0M will not use any encryption to connect to the specified network.
wep	The MRF24WB0M / MRF24WG0M will use either WEP-40 (short) or WEP-104 (long) encryption to connect to the specified network.
wpa-psk	The MRF24WB0M / MRF24WG0M will use the specified 32-byte PSK to connect to the WPA-PSK/WPA2-PSK network.
wpa-phrase	The MRF24WB0M / MRF24WG0M will take the given 1-32 ASCII character passphrase, along with the SSID, and compute the required 32-byte PSK for the network. Note that doing so takes approximately 30 seconds to complete the calculation.

key	
[1]	Instructs the MRF24WB0M / MRF24WG0M to use this key for connecting to the WEP encrypted network. Note that only key 1 is considered safe to use among different AP vendors. Keys 2-4 can have implementation specific entries that may not be compatible from AP to AP, and are not supported.

value	<p>If value is specified, this will instruct the MRF24WB0M / MRF24WG0M to use the specified key number and also program the device with this key value. For WEP-40 networks, this implies the key is either 5 ASCII characters or 10 hex characters in length. For WEP-104 networks, this implies the key is either 13 ASCII characters or 26 hex characters in length.</p> <p>The console only accepts hex WEP keys. Therefore, the user must do the ASCII to hex conversion for their ASCII keys.</p>
psk	
value	<p>32-byte hex value for the PSK.</p> <p>This value can be calculated from the following website hosted on the Wireshark website.</p>
phrase	
value	<p>An 8-63 ASCII character phrase (delimited with quotes if using spaces). This phrase will be used along with the SSID to generate the 32-byte PSK value for the network.</p>

1.6.2.2.1.4 iperf Example

An example of using iperf to measure network bandwidth and performance.

Description

iperf is a networking tool that helps to measure networking bandwidth and performance. The console demo application has a built-in iperf application, that can act as both a client and server for testing. iperf has the ability to test both UDP and TCP. In the case of UDP, you can specify the size of the UDP datagrams. For TCP, iperf measures the throughput of the payload.

In order to run iperf, you'll need a PC that has an iperf application on it as well. There is an open source version that is maintained, as well as many other variants across the internet. iperf is meant to be run at the command line. However, if a GUI is desired, a variant called jperf can be used.

In the case of the demo application, iperf measures performance data in a unidirectional format. Therefore, the side that the server is running on is considered the receiver, and provides the most accurate performance values.

Command Synopsis

iperf	[-s -c <IP addr>] [-u] [-i <sec>] [-b <bandwidth>] [-t <time>]
-s	Runs the iperf server. No IP address needs to be specified.
-c <IP addr>	Runs the iperf client. The IP address is the IP address of the server.
-u	Server side only. Sends UDP datagrams.
-i <sec>	Specified the time interval, in seconds, that the display will be updated.
-b <bandwidth>	Specifies the amount of data to try and send. This option is only valid with UDP datagrams.
-t <time>	Amount of time, in seconds, to run the test.

Running the Demo

After powering on the development board and associating with your wireless network, you'll need to start the server side iperf application first. If you start iperf as a server on the development board in the console, then this implies that you are trying to measure the MRF24WB0M / MRF24WG0M receiver performance. If you start the iperf server on a PC, then you will be measuring MRF24WB0M / MRF24WG0M transmit performance. Below are two images that show receiver and transmitter performance, respectively.

1.6.2.3 TCPIP WiFi Demo App

Main demo application for the TCP/IP Stack.

Description

The **TCPIP Demo App** project folder contains the main demo application for the Microchip TCP/IP Stack. Besides showing example applications using the web server, e-mail client, SNMP server, and more, this application also includes examples for implementing custom application layers. Details about these applications are provided here.

For a list of pre-tested demo hardware configurations, please consult the Demo Compatibility Table. Unspecified hardware configurations may also be useable with the Demo App, but some additional configuration may be necessary.

Some demo features are disabled in certain Demo App projects to support the associated hardware platform and TCP/IP controller. Please consult the following table to determine which features are available on which configurations:

1.6.2.3.1 TCPIP WiFi Demo Modules

Describes the custom modules used in this demo.

Description

Several custom modules are used in this demo. This section will describe the components and functionality of these modules.

For Microchip 802.11b/g WiFi PICtail MRF24W, the following network types are supported.

- CFG_WF_INFRASTRUCTURE
- CFG_WF_ADHOC
- CFG_WF_P2P (supported by MRF24WG0M only)

For Microchip 802.11b/g WiFi PICtail MRF24W, the new security modes are supported.

- WF_SECURITY_WPS_PUSH_BUTTON (supported by MRF24WG0M only)
- WF_SECURITY_WPS_PIN (supported by MRF24WG0M only)

1.6.2.3.1.1 Web Page Demos

Provides an example for building a custom HTTP application using the HTTP2 server and allows several other demo features to be accessed and controlled via web interface.

Description

The custom_http_app.c file demonstrates how to build a custom HTTP application on top of the HTTP2 server. All the features of the TCPIP WiFi Demo App web pages are implemented here. Examples can be found for handling Authentication, processing web forms (using HTTP GET and POST), and providing status information through the output of dynamic variables.

1.6.2.3.1.1.1 Dynamic Variables

Dynamic variables allow the PIC to output custom data when serving a web page.

Description

Overview

This section describes how to view dynamic variables in the TCP/IP Demo App HTTP2 demo. For information about how to implement dynamic variables in your own page, see the HTTP2 dynamic variables topic.

Instructions

1. Program your board with the demo code and upload the demo app web page. Open your web browser and navigate to the board's web page (<http://mchpboard> by default).
2. Observe the LED output, button state, and potentiometer reading in the box in the upper right of the web page. The button and potentiometer values will be updated dynamically based on the status of the buttons on your board. In addition, if you click the LEDs to toggle them, their status will be dynamically updated on the page. Note that some LEDs or buttons may not be implemented, depending on your hardware setup. Consult the TCPIP Demo App Features by Hardware Platform topic for more information.
3. Observe the current Stack Version and Build Date in the top center of the Overview Page.
4. Navigate to the Dynamic Variables page using the navigation panel on the left of the page.
5. Observe the Build Date and Time, LED state, stack version, and current IP address- these variables are output to this page dynamically when it's downloaded by the browser.

Exercises

You can optionally complete the exercises described on the Dynamic Variables page. You may want to read the HTTP2 dynamic variables topic first. The first exercise is to implement the display of LED0 on the dynamic variable demo page.

1. Start by opening `dynvars.htm` in your "TCPIP Demo App\WebPages2" folder.
2. Locate the dynamic variables in the page and replace the question mark with a dynamic variable to display the value of LED 0. You can use the other LED variables as a template, but specify 0 as the LED to open.
3. In your MPLABX project, open `custome_http_app.c` and ensure that the `HTTPPrint_led` function (if you used `~led(0)~` as your dynamic variable) is written to output data when 0 is passed in as a parameter.
4. Rebuild your web page with the MPFS2 Utility.
5. Rebuild your project, and reprogram your board. Navigate to the dynamic variable page and verify that the LED0 field reflects the status of the LED on your board. Since the LED on your board is blinking, you may need to refresh the web page to view its current status.

The second exercise on this page simply demonstrates how to dynamically insert a file into a web page.

1. Start by opening `dynvars.htm` in your "TCPIP Demo App\WebPages2" folder.
2. Locate the dynamic variables that include `header.inc` and `footer.inc`. Observe the difference between the declaration of these variables and the other variables on the page.

1.6.2.3.1.1.2 Authentication

Authentication requires a correct username/password prompt before displaying select web pages.

Description

Overview

This section describes how to use the authentication demo in the TCP/IP Demo App HTTP2 demo. For information about how to implement authentication in your own page, see the HTTP2 Authentication topic.

Instructions

1. Program your board with the demo code and upload the demo app web page. Open your web browser and navigate to the board's web page (<http://mchpboard> by default).
2. Navigate to the Authentication page using the navigation panel on the left of the page.

3. Note the authentication user name ("admin") and password ("microchip").
4. Click on the "Access Restricted Page" link on the Authentication page.
5. Enter an incorrect combination of usernames and passwords. The browser will not advance to the Access Restricted Page. After 3 incorrect username/password combinations, the browser will be redirected to an "Unauthorized" screen.
6. Click the back button in your browser. Click on the "Access Restricted Page" link and enter the correct username and password.
7. You will advance to the "Login Successful" page. Your browser will store this username/password combination until it is closed and reopened.

Exercise

You can optionally complete the exercise described on the "Login Successful" page. In this exercise, you will change the username and password that you use to log in to this page.

1. Open `custom_http_app.c` in your TCP/IP Demo App MPLABX project.
2. Locate the `HTTPCheckAuth` function.
3. Change the values being compared to the function inputs to a username and password of your choosing.
4. Rebuild your project and program your board.

1.6.2.3.1.1.3 Forms using GET

Web forms are easier to process when using the GET method, but the amount of data is limited.

Description

Overview

This section describes how to use web forms in the TCP/IP Demo App HTTP2 demo. For information about how to implement forms in your own page, see the HTTP2 form processing topic.

Instructions

1. Program your board with the demo code and upload the demo app web page. Open your web browser and navigate to the board's web page (`http://mchpboard` by default).
2. Observe the LED state on the board. Click on an LED indicator in the box on the top right of the Overview page. Verify that the LED state changes on the board. Note that some LEDs or buttons may not be implemented, depending on your hardware setup. Consult the TCPIP Demo App Features by Hardware Platform topic for more information.
3. Navigate to the Form Processing page using the navigation panel on the left of the page.
4. Select new LED states in the pull-down boxes. Click "Save" and observe that the LED states of your board changed to match the settings you selected.

Exercise

You can optionally complete the exercise described on the "Form Processing" page. In this exercise, you will change the example to support LED5. You may want to read the HTTP2 form processing topic first.

1. Start by opening `forms.htm` in your "TCPIP Demo App\WebPages2" folder.
2. Locate the GET method implementation that will display the LEDs. You should see `select` forms for the four LEDs that are already implemented. Each of these has two options: the On option will send a '1' to the server when submitted and the Off option will send a '0' when submitted. Each of the declarations of these options also use the `ledSelected` dynamic variable to determine which option will be selected by default, based on the current status of the corresponding LED on the board. This dynamic variable accepts two arguments: the first defines which LED is being checked, and the second describes the state being checked for. So, for example, the `~ledSelected(4,TRUE)~` variable will be replaced by the word "SELECTED" if LED4 is on when this variable callback function is called. In this case, `~ledSelected(4,FALSE)~` would be replaced by nothing. This would result in the 'On' option being selected by default in the page.
3. Create a new `select` input for LED5.
4. Open `custom_http_app.c` in the TCP/IP Demo App MPLABX project.
5. Verify that the `HTTPPrint_ledSelected` dynamic variable callback function has been implemented for LED5.
6. Find the `HTTPExecuteGet` function. Locate the section of code that processes GET arguments for the `forms.htm` file.

7. Add implementation to search for the "led5" argument string in the GET data buffer and then set LED5_IO based on the associated value.

1.6.2.3.1.1.4 Forms using POST

Forms using POST are more difficult to process, but large amounts of data can be submitted.

Description

Overview

This section describes how to use web forms in the TCP/IP Demo App HTTP2 demo. For information about how to implement forms in your own page, see the HTTP2 form processing topic.

Instructions

1. Program your board with the demo code and upload the demo app web page. Open your web browser and navigate to the board's web page (<http://mchpboard> by default).
2. Navigate to the Form Processing page using the navigation panel on the left of the page.
3. Enter a text string into the "LCD" text box and click on "Save." Verify that this string was written to the LCD display on your demo board.
4. Navigate to the File Uploads page using the navigation panel on the left of the page.
5. Browse for a file on your computer and click "Get MD5." The application will read your file using a series of POST transfers and calculate and display and MD5 hash of the contents.
6. Navigate to the Send E-mail page using the navigation panel on the left of the page.
7. Fill in the form fields with the appropriate information.
 1. No SSL - You will need a local SMTP server that does not require a secure connection. Enter the address in the SMTP Server field, set the port to 25, and enter your user name and password for the server. Set the "To:" field to the email recipient and press "Send Message."
 2. SSL - Enter the address of a public SMTP server (e.g. smtp.gmail.com). Set the port number to 465 or 587. Enter your email account information (e.g. username@gmail.com and your Gmail password). Set the "To:" field to the email recipient and press "Send Message." Note that some corporate subnets may block outgoing secure traffic on the SMTP port. If this is the case, you'll have to establish a VPN tunnel outside this network or connect your board to a network that's not blocked by this type of firewall. You must have installed the Microchip Data Encryption Libraries to use SSL, and SSL Client support must be enabled. See the SSL API topic for more information.
8. Verify that the e-mail was received on the recipient e-mail address.

1.6.2.3.1.1.5 Cookies

Cookies allow storage of session information in a user's browser.

Description

Overview

This section describes how to use the cookie demo in the TCP/IP Demo App HTTP2 demo. For information about how to implement cookies in your own page, see the HTTP2 Cookies topic.

Instructions

1. Program your board with the demo code and upload the demo app web page. Open your web browser and navigate to the board's web page (<http://mchpboard> by default).
2. Navigate to the Cookies page using the navigation panel on the left of the page.
3. Type your first name into the "First Name" text box and click "Set Cookies." Verify that the name was read successfully and displayed in the "Name" output field.

Exercise

You can optionally complete the exercise described on the "Cookies" page. In this exercise, you will create a cookie called "fav" with the value in the favorite field in the example box. You may want to read the HTTP2 dynamic variable, GET, and cookie topics first.

1. Start by opening `cookies.htm` in your "TCPIP Demo App\WebPages2" folder.
2. Locate the code for the example box that displays the name and favorite PIC architecture. Replace the "not implemented" string with a dynamic variable to output the data from the cookie.
3. Locate the code for the select box form input for the favorite architecture. Note the value of the name field of the select form.
4. Open `custom_http_app.c` in the TCP/IP Demo App MPLABX project. Locate the `HTTPExecuteGet` function and find the code that handles GET method inputs from `cookies.htm`.
5. Set the value of `curHTTP.hasArgs` to indicate that two form arguments are present in the data buffer.
6. In `custom_http_app.c`, create a function to output data for the dynamic variable you created in step 2. The name of the function will depend on the name of the variable. For a variable named `~cookiefavorite~` you would implement a function called `HTTPPrint_cookiefavorite`. This function should search through the `curHTTP.data` data buffer to try and find a name/value pair with the name equal to the name of your select form from step 3. If it finds it, it should write the value for that pair to the TCP buffer; otherwise, it should write "not set." See the implementation of `HTTPPrint_cookiename` for an example.

```
void HTTPPrint_cookiename(void)
{
    BYTE *ptr;

    ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE*)"name");

    if(ptr)
        TCPPutString(sktHTTP, ptr);
    else
        TCPPutROMString(sktHTTP, (ROM BYTE*)"not set");

    return;
}
```

7. Compile your web page using the MPFS2 Utility and upload it to your board. You may receive a warning that your dynamic variables have changed in your page.
8. Rebuild your project and program your board.
9. Verify that both cookies can be set.

1.6.2.3.1.2 E-mail (SMTP) Demo

Demonstrates how to use an e-mail client to send messages when events occur. This is a standalone demo; for the web page "Send Email" demo, see the Forms using POST topic.

Description

Overview

This file provides two examples for using the SMTP client module to send e-mail messages. The first transmits short alert messages whose entire bodies can be stored in RAM at once. The second example demonstrates how to generate messages on-the-fly when the entire body cannot be allocated in RAM. (This second example is commented. You must comment the first example and uncomment this one to use it.)

A third example of using the SMTP client is provided in `HTTPPostEmail`. This example shows how to send messages with attachments, as well as how to dynamically configure the recipient and e-mail server at run-time.

Instructions (Short Message Demo)

1. Open your project in MPLABX and open `SMTPDemo.c`. Scroll down to the `MAIL_BEGIN` case in the switch statement in the `SMTPDemo()` function.
 1. Replace the initializer of the `RAMStringTo[]` array with the target email address.
 2. Replace the initializer of the `SMTPClient.Server.szROM` structure element with the address of your mail server. Note that this demo does not include security features, so you will need a mail server that does not require SSL. To test this functionality with a mail server that does support SSL (including most public mail servers), please use the `HTTPPostEmail` SMTP demo.
2. Compile the code, program your board, and run the demo.

3. Press buttons 2 and 3 on your board to transmit an email message. LED1 on your board will indicate that the message is being transmitted; LED2 will indicate that it was sent successfully. Check the `BUTTON2_IO`, `BUTTON3_IO`, `LED1_IO`, and `LED2_IO` macros in the copy of `system_config.h` and `system.h` that correspond to your project to determine which buttons and LEDs are used for your hardware setup.
4. Verify that the message was received by the email account you specified in the `RAMStringTo[]` array.

Description

The short-message SMTPDemo task function implements a four-state state machine. When the board is powered on, the state machine is initialized to the `SM_HOME` state, in which it waits for buttons 2 and 3 to be pressed. Once they are pressed, the task will enter the `MAIL_BEGIN` state.

In the `MAIL_BEGIN` state, the task will attempt to requisition the SMTP module. Once it's able to do this, it will populate the `SMTPClient` structure with message parameters and transmit the message. It will then enter the `MAIL_SMTP_FINISHING` state.

In the `MAIL_SMTP_FINISHING` state, the task will check a callback function (`SMTPIsBusy`) to determine when the module is finished. It will then give up control of the SMTP module and toggle LEDs based on the successful operation of the SMTP module. The state machine will then enter the `MAIL_DONE` state, which will wait at least 1 second before transitioning back to `MAIL_HOME`, allowing another email to be sent.

1.6.2.3.1.3 Generic TCP Client

Demonstrates how to build a TCP Client application through an HTTP client example.

Description

Overview

The Generic TCP Client provides an example of how to build an HTTP client (or any other TCP client) using the Microchip TCP/IP Stack. It will print out the results from a search engine query to the PIC's UART module. The result data can be viewed on a PC terminal.

Instructions

1. Connect the programmed demo board to a router that is connected to the Internet.
2. Connect your PC to your demo board with an RS-232 cable. Open a terminal program like HyperTerminal, and configure it to the following settings: 19200 bps, 8 data bits, No parity, 1 stop bit, No flow control.
3. Press Button 1 on your demo board (check the `BUTTON1_IO` macro in the copy of `system_config.h` and `system.h` that correspond to your project to determine which button is Button 1).
4. Observe the search results for "Microchip" at www.microchip.com on your terminal.

Description

The Generic TCP Client demo implements a task function with five states. When the board is powered on, the initial state will be set to `SM_DONE`. This state will wait for the user to press Button 1; when a button-press event occurs, the state will switch to `SM_HOME`. In the `SM_HOME` state, the task will attempt to open a TCP client socket. This socket will use a `TCP_PURPOSE_GENERIC_TCP_CLIENT` socket type from the TCP socket structure that was initialized in your configuration files. The targeted server will be the Google search engine, and the server port will be 80, the port used for HTTP connections. The task will switch the state machine to the `SM_SOCKET_OBTAINED` state.

The task will wait in the `SM_SOCKET_OBTAINED` state until a connection is established with Google or a 5-second timeout elapses. If a timeout occurs, the state will close the socket and change the state back to `SM_HOME`. Otherwise, it will wait until the TCP buffer can accept 125 bytes of data and then use an HTTP GET to search for the word "Microchip" at the site "microchip.com." Once the GET has been sent, the state will switch to `SM_PROCESS_RESPONSE`.

In the `SM_PROCESS_RESPONSE` state, the task will wait until a response is received or the socket was disconnected. If a response is received, it will print it to the UART. In either case, the task will transition to the `SM_DISCONNECT` state, where it will close the client socket and return to the `SM_DONE` state.

1.6.2.3.1.4 Generic TCP Server

Demonstrates how to build a TCP server application

Description

Overview

The Generic TCP Server example demonstrates how to build a TCP server application. Once you connect to the demo server, it will echo your keystrokes back to you after converting the characters to UPPER CASE.

Instructions

1. Connect the programmed demo board to a computer either directly or through a router. For Ethernet, a direct connection may require a crossover cable; for WiFi, the board may need to be in AdHoc mode to establish a direct connection.
2. Determine the IP address of the demo board. This can be done several different ways.
 1. If you are using a demo setup with an LCD display (e.g. Explorer 16), the IP address should be displayed on the second line of the display.
 2. Open the Microchip TCP/IP Discoverer from the start menu. Press the "Discover Devices" button to see the addresses and host names of all devices with the Announce Protocol enabled on your network. You may have to configure your computer's firewall to prevent it from blocking UDP port 30303 for this solution.
 3. If your board is connected directly with your computer with a crossover cable:
 1. Open a command/DOS prompt and type 'ipconfig'. Find the network adaptor that is connected to the board. The IP address of the board is located in the 'Default Gateway' field
 2. Open up the network status for the network adaptor that connects the two devices. This can be done by right clicking on the network connection icon in the network settings folder and select 'status' from the menu. Find the 'Default Gateway' field.
3. Open a command/DOS prompt. Type "telnet ip_address 9760" where ip_address is the IP address that you got from step 2 and 9760 is the TCP port chosen for the Generic TCP Server implementation.
4. As you type characters, they will be echoed back in your command prompt window in UPPER CASE.
5. Press Escape to end the demo.

Description

The GenericTCPServer demo implements a task function with 3 states. In the first state, SM_HOME, the task will attempt to open a TCP server socket. This socket will use a TCP_PURPOSE_GENERIC_TCP_SERVER socket type from the TCP socket structure that was initialized in your configuration files. It will also listen on TCP port 9760 (defined by the macro SERVER_PORT).

Once the socket has been successfully opened, the task function will enter the SM_LISTENING state. In this state, the task will always return unless a client has connected to it (by establishing a telnet connection on port 9760). Once a client has connected to the server, the server will read received data from the TCP socket's RX buffer, convert it to upper case, and write it to the TCP socket's TX buffer.

If an Escape character is received, the server will enter the SM_CLOSING state. In this state, it will close the server socket to break the current connection. The server will then re-enter the SM_HOME state, where it will reopen the TCP_PURPOSE_GENERIC_TCP_SERVER socket to listen for new connections.

1.6.2.3.1.5 Ping (ICMP) Demo

Demonstrates how to build a Ping client.

Description

Overview

The Ping Demo explains how to use the ICMP client to check if a remote node is reachable. If the project with this demo includes the DNS module, the PIC will ping "ww1.microchip.com." Otherwise, it will ping the local gateway. This demo is only available on hardware setups with LCD displays (e.g. Explorer 16).

Instructions

1. Press Button 0 on your demo board. Button 0 is usually the rightmost or topmost button on the board (check the `BUTTON0_IO` macro in the copy of `system_config.h` and `system.h` that correspond to your project to determine exactly which button is Button 0).
2. When the device receives an echo response from the remote node or when the ping times out, the LCD will be updated with the appropriate information.

Description

The PingDemo task function implements a two-state state machine. The task will wait in the `SM_HOME` state until the user presses button 0. Once the button is pressed, the task will attempt to obtain ownership of the ICMP module with the `ICMPBeginUsage` function. If it does, it will send a ping to the specified address and transition to the `SM_GET_ICMP_RESPONSE` state.

In the `SM_GET_ICMP_RESPONSE` state, the task will call the `ICMPGetReply` callback function and take action depending on the return value:

Value	Action
-2	Remain in this state and keep waiting for a response.
-1	Write a message to the LCD indicating that the ping timed out. Change state to <code>SM_HOME</code> .
-3	Write a message to the LCD indicating that the DNS module couldn't resolve the target address. Change state to <code>SM_HOME</code> .
Other	Convert the response time to a text string and print it to the LCD. Change state to <code>SM_HOME</code> .

1.6.2.3.1.6 UART-to-TCP Bridge

Describes how to use the UART-to-TCP Bridge demo.

Description

Overview

The UART-to-TCP bridge feature of the TCP/IP Demo App transmits all incoming TCP bytes on a socket out of the PIC's UART module and all incoming UART bytes out of a TCP socket.

Instructions

1. Compile your MPLABX project and program the demo board.
2. Connect the RS-232 port on your demo board to an RS-232 port on your computer. On a newer computer you may need an RS-232 to USB converter cable. On your computer, open a terminal program (such as HyperTerminal). Set it to use the COM port you connected your board to, at 19200 baud, with 8 data bits, no parity, 1 stop bit, and no flow control.
3. Connect the programmed demo board to a computer either directly or through a router. For Ethernet, a direct connection may require a crossover cable; for WiFi, the board may need to be in AdHoc mode to establish a direct connection.
4. Determine the IP address of the demo board. This can be done several different ways.
 1. If you are using a demo setup with an LCD display (e.g. Explorer 16), the IP address should be displayed on the second line of the display.
 2. Open the Microchip Ethernet Device Discoverer from the start menu. Press the "Discover Devices" button to see the addresses and host names of all devices with the Announce Protocol enabled on your network. You may have to configure your computer's firewall to prevent it from blocking UDP port 30303 for this solution.
 3. If your board is connected directly with your computer with a crossover cable:
 1. Open a command/DOS prompt and type 'ipconfig'. Find the network adaptor that is connected to the board. The IP address of the board is located in the 'Default Gateway' field
 2. Open up the network status for the network adaptor that connects the two devices. This can be done by right clicking on the network connection icon in the network settings folder and select 'status' from the menu. Find the 'Default Gateway' field.
5. Open a command/DOS prompt. Type "telnet ip_address 9761" where ip_address is the IP address that you got from step 4.

6. As you type characters in the command prompt, they will be transmitted over the Telnet TCP port to the PIC, and then transmitted out of the PIC's UART to appear on your terminal program. As you type characters in the terminal program, they will be transmitted to the PIC through the UART module, and then retransmitted over the TCP connection to appear in the command prompt Telnet session.

1.6.2.3.1.7 Zero Configuration (ZeroConf)

Describes usage of the Zeroconf modules

Description

Zero configuration (Zeroconf), provides a mechanism to ease the configuration of a device on a network. It also provides for a more human-like naming convention, instead of relying on IP addresses alone. Zeroconf also goes by the names Bonjour (Apple) and Avahi (Linux), and is an IETF standard.

Enabling

Zeroconf can be enabled by setting the following two defines in `tcpip_config.h`:

- `STACK_USE_ZEROCONF_LINK_LOCAL`
- `STACK_USE_ZEROCONF_MDNS_SD`

Currently, the use of Zeroconf is limited to the WiFi demo applications (and the MRF24WB0M / MRF24WG0M module). Future versions of the stack should enable Zeroconf support across all Ethernet solutions.

Link Local

The first component of Zeroconf is the ability to self-assign an IP address to each member of a network. Normally, a DHCP server would handle such situations. However, in cases where no DHCP server exists, Zeroconf enabled devices negotiate unique IP addresses amongst themselves.

mDNS

The second component of Zeroconf is the ability to self-assign human-readable hostnames for themselves. Multicast DNS provides a local network the ability to have the features of a DNS server. Users can use easily remembered hostnames to access the devices on the network. In the event that devices elect to use the same hostname, as in the IP address resolution, each of the devices will auto-negotiate new names for themselves (usually by appending a number to the end of the name).

Service Discovery

The last component of Zeroconf is service discovery. All Zeroconf devices can broadcast what services they provide. For instance, a printer can broadcast that it has printing services available. A thermostat can broadcast that it has an HVAC control service. Other interested parties on the network who are looking for certain services can then see a list of devices that have the capability of providing the service, and connect directly to it. This further eliminates the need to know whether something exists on a network (and what its IP or hostname is). As an end-user, all you would need to do is query the network if a certain service exists, and easily connect to it.

Demo

The demo, when enabled, shows all three items above working together. Each development kit in the network assumes the hostname of `MCHPBOARD-x.local`, where `x` is an incrementing number from 1 (only in the case where multiple kits are programmed for the network). Each board will broadcast its service, which is the DemoWebServer.

Zeroconf Enabled Environments

All Apple products have Zeroconf enabled by default. On Windows, you'll need to download the Safari web browser, and during the install, enable support for Bonjour. Note that in the Safari browser, you can browse and see a list of all Bonjour enabled devices, and click through to them automatically.

1.6.2.4 TCPIP WiFi EasyConfig Demo

Implements a wireless application to configure an embedded device for a network in which the device has no natural human interface mechanism (i.e. keyboard and display). This demo also demonstrates dynamic scanning of networks and getting back calculated PSK keys from the MRF24W device.

Description

Overview

WLAN networks provide a unique challenge for configuring embedded wireless without a natural user interface. Unlike wired networks, wireless networks require unique items such as the SSID and network type and keys, which have to be sent to the device in some form or another. Traditionally, this means a user would enter this information using a keyboard and display.

EasyConfig is a mechanism to allow for configuration of an embedded device on a wireless network. It utilizes the web server of the TCP/IP stack, as well as a wireless adhoc (IBSS) network to allow the user to input the desired network information from a client browser, and then reset the device to connect to the desired network.

The EasyConfig demo works in roughly the following manner:

1. Upon power up the device, it broadcasts an adhoc network with SSID "EasyConfig".
2. A client device (laptop, iPod Touch/iPhone/iPad) can then connect to the EasyConfig network.
3. Upon connecting, the user can then use a standard web browser to go to the IP address of the demo (<http://169.254.1.1>).
4. The user will then be presented with some web pages from the web server. The index.htm web page has some additional information on EasyConfig, and also shows the continually updating status of the LEDs, buttons, and potentiometer on the development board. The configure.htm page will allow the user to scan for networks, and connect to a network of their choosing.
5. The device will then reset itself, using the parameters for the new network. In order to continue using the demo, the client device will now need to reconnect to the same network that the development board is on.

Note that the demo will always attempt to connect to the last known network. If the user wants to reset the demo to startup in adhoc mode again, then button S3 on the Explorer 16 development board needs to be held down for 4 seconds.

The following network types are supported

- CFG_WF_ADHOC
- CFG_WF_SOFT_AP

Wireless configurations are set up in `drv_wifi_config.h`

Adhoc Networks

Upon starting the demo, the network will either connect to another adhoc network, or will start its own if one is not found. Adhoc networks are peer-to-peer networks, with no centralized coordinator for the network. All the devices in the network share the responsibilities of keeping the network going.

One downfall of adhoc networks is that typically security is not employed on them. The MRF24WB0M / MRF24WG0M module can secure the network with WEP (40-bit/104-bit) security, as can most laptops and adhoc devices. Almost no devices in the market can secure an adhoc network with WPA level security due to the tremendous overhead in doing so.

The demo starts an adhoc network with no security. This means that all the network information that is being configured on the device is going over-the-air in the open. For most applications, unless somebody is specifically attempting to eavesdrop on this network, there should be little to no impact on security. However, for applications that do require some baseline level of security, then WEP can be employed on the network. SSL can also be used to encrypt the traffic between the web server and client browser. Additionally, some other form of data-level security can be employed to obfuscate the ASCII network information being sent to the device.

SoftAP Networks

This is only available for MRF24WG0MA/B. Upon starting the demo, the MRF24WG0MA/B will start up a network as a software-enabled access point (AP), acting as the centralized coordinator for the network. Devices can connect to the MRF24WG0MA/B softAP. Depending on the RF module firmware version, either 1 or 4 clients can be connected to the softAP. Routing is not supported. The demo can start a softAP network with no or WEP security.

Network Parameters

Below is some information on the parameters that are being sent via HTTP POST from the client browser to the device. All this information is being parsed and handled in the function `HTTPPostWifiConfig()` in `custom_http_app.c`.

WLAN Type	Either adhoc or infrastructure
SSID	Name of network (1-32 ASCII characters)
Security Type	<ul style="list-style-type: none"> • None • WEP-40 (5 ASCII characters or 10 hex numbers) • WEP-104 (13 ASCII characters or 26 hex numbers) • WPA-PSK/WPA2-PSK passphrase (8-63 ASCII characters)

Configured vs Un-configured State

When the demo is running in an unconfigured state (i.e, serving the default EasyConfig SSID in adhoc mode), then the heartbeat LED (LED0) will blink twice per second to indicate that it hasn't been configured yet. Once the network has been configured, then the heartbeat LED will change to blink once per second, in a similar fashion to the other TCP/IP demo applications.

EasyConfig Demo Additional Features

There are four defines that enable EasyConfig as well as extend it with natural features.

<code>STACK_USE_EZ_CONFIG</code>	The top level define to enable EasyConfig features.
<code>EZ_CONFIG_SCAN</code>	Adds additional ability to instruct the MRF24WB0M / MRF24WG0M module to scan for available networks nearby. This can be done when you are already connected to a network.
<code>EZ_CONFIG_STORE</code>	Store the configuration information for the new network to non-volatile memory. In the event that WPA-PSK/WPA2-PSK level security is used, the 32-byte PSK will be saved to NVM.
<code>EZ_CONFIG_STALL</code>	Before switching networks, forces the configuration state machine to pause. This gives the client device additional time to request resources from the development platform before it attempts to connect to a new network.

EZ_CONFIG_SCAN

The MRF24WB0M / MRF24WG0M has the ability to scan for nearby networks. This is similar to a laptop that can show available wireless networks that can be connected to. The scan results are stored on the MRF24WB0M / MRF24WG0M module, and can be retrieved one at a time from the device. This helps to reduce the impact of storing all the scan results on the host itself.

The scan can be performed when idle, or when connected to either an adhoc or infrastructure network.

EZ_CONFIG_STORE

The new network parameters can also be stored to non-volatile memory. For the Explorer 16 development board, this information is stored to the 32kB EEPROM on the board.

One extremely useful feature of storing the information surfaces when connecting to a network with WPA-PSK/WPA2-PSK security. The computation of the 32-byte PSK is computationally heavy, and can take the MRF24WB0M / MRF24WG0M up to 30 seconds to calculate the key. In a normal application, it would be unacceptable to have to wait 30 seconds every time the device started up before connection to the network was established.

`EZ_CONFIG_STORE` helps to alleviate doing the calculation each time by storing the 32-byte PSK to NVM. In doing so, there is only one 30-second hit the very first time the key is calculated only. Successive connections to the network will be significantly faster.

EZ_CONFIG_STALL

The configuration state machine that controls the network connections within EasyConfig can employ a wait state between switching networks. From an end user experience, this becomes vital. If the switch between different networks was instantaneous, a client browser would never get an indication that the HTTP session was closed after the POST information was sent. The end user would see this as a browser that was continually waiting, which would eventually timeout.

To make the switch more natural and complete, `EZ_CONFIG_STALL` adds additional time to allow the client to get the remaining web page information. For the demo, this includes a HTTP redirect to a page that highlights the new network information.

Current Incompatibilities

The javascript being used in EasyConfig is not compatible with Internet Explorer 7. EasyConfig does work with many other flavors of browser on different architectures, not limited to Internet Explorer 8, Mozilla Firefox, Apple Safari and Google Chrome. The incompatibility is something that is being investigated, and should be fixed in a future stack release.

1.7 Using the Stack

Describes how to use the TCP/IP Stack.

Description

This section describes how to use Microchip's TCP/IP Stack.

1.7.1 Stack Architecture

Describes the TCP/IP stack architecture.

Description

The TCP/IP stack is modular in design and written in the 'C' programming language. It follows the TCP/IP (Internet) protocol suite. The stack currently supports the TCP and UDP transport layer modules, the IPv4 (and part of the ICMP) Internet Layer modules, the ARP link layer modules, and a variety of application layer modules. Most of the Media Access Control link layer functionality is provided by the hardware MAC/PHY chips used with the stack.

1.7.2 How the Stack Works

Describes how the stack works and how to start creating your application.

Description

This topic contains information about how the stack works, what is required to use the stack, and how your code can be structured to work cohesively with the TCP/IP stack.

1.7.2.1 Required Files

Describes the base files needed in a TCP/IP project.

Description

There are several base files that must be included in every project using Microchip's TCP/IP stack. They are:

- A main file- this is the file with your application code in it..
- `arp.c` and `arp.h`- These files are used by the stack to discover the MAC address associated with a given IP address.
- `delay.c` and `delay.h` – These files are used to provide delays for some stack functions. Note that it would be best to not use these delays in your own code, as they do create blocking conditions.
- Physical layer files – These files are used to enable a specified physical layer. More information on which files to include can be found in the Hardware Configuration section.
- `helpers.c` and `helpers.h` – These files contain helper functions used for miscellaneous stack tasks.
- `ip.c` and `ip.h` – These files provide internet layer functionality for the stack.
- `stack_task.c` and `stack_task.h` – These files contain the code to initialize the stack and perform the callbacks that keep the stack going.
- `tick.c` and `tick.h` – These files implement a tick timer that is used to implement some timing functionality within the stack.
- `system_config.h` – This configuration file is used to set up hardware options.

- `tcp_config.h` – This configuration file is used to set up firmware options.
- `mac.h` – This header file provides macros and structures relating to the hardware MAC layer.
- `tcPIP.h` – This is the primary include file for the stack. Your main file should include `tcPIP.h`.

You may choose to include additional files to support additional protocols and features. The list of protocols and their required files can be found in the Protocol Macros and Files topic in the Protocol Configuration topic.

1.7.2.2 APP_CONFIG Structure

Describes the APP_CONFIG structure.

Description

Most of the stack-related application variables are stored in an APP_CONFIG structure. These include addresses, flags, and NBNS/SSID name strings. You will need to declare one of these structures (named "AppConfig") for your application and initialize it with the default values defined in `tcPIP_config.h`. For example, you would set the bytes of the `MyIPAddr` field to the values of the `MY_DEFAULT_IP_ADDR_BYTEn` macros in `tcPIP_config.h`. The `InitAppConfig` function in the file `main.c` of the TCPIP Demo App project demonstrates how to populate this structure completely. The full list of parameters in the APP_CONFIG structure is defined in the file `stack_task.h`.

At the beginning of most stack demonstration applications, the code will check an EEPROM to determine if it contains a valid image of an APP_CONFIG structure. If so, it will read the image and use it to populate the AppConfig instance in the demo project. Otherwise, it will load the application variables from your statically defined values and/or configure them based on application protocols (DHCP/AutoIP). This allows a board to retain its configured settings even if the application loses power.

1.7.2.3 Main File

Describes how to construct the `main()` loop in your application.

Description

Because there is a huge variety of ways in which you could write your application, this section will provide an outline of what your main file should contain. It also provides some description of the stack operation, and of best-practice programming techniques to prevent stack problems.

1.7.2.3.1 Initialization

Describes some basic initialization calls that should be made.

Description

You should start by initializing your hardware. This includes PPS pins, oscillators, LEDs, LCDs, any SPI or PMP modules you're using to control your hardware MAC/PHY chip, etc.

Next, call the hardware initialization functions for the library. `TickInit()` should be called first; it will initialize the tick timer that manages your stack timing. Then call any additional initialization functions that require hardware initialization. For example, the `MPFSInit()` function will need to initialize an SPI port to communicate to a memory storage device to store web pages, so it should be called now.

Once your hardware is initialized, you can begin configuring your stack. Most of the stack-related application variables are stored in the AppConfig structure. At this point, you should initialize the AppConfig structure with your default values, or provide another means of initializing the AppConfig structure.

Finally, you can initialize the stack by calling the `StackInit()` function. This function will automatically call the initialization functions for other firmware protocols if they have been enabled in `tcPIP_config.h` (i.e. `TCPInit()` for the TCP protocol, `HTTPInit()` for HTTP2,...). After `StackInit()` has been called, you can call other application-specific firmware initialization functions.

1.7.2.3.2 Main Loop

Describes how to construct the main loop of your application.

Description

Once your program has been initialized, you should enter an infinite loop which will handle your application tasks. Within this loop, there are two functions that you must call regularly: `StackTask` and `StackApplications`.

The `StackTask` function will perform any timed operations that the stack requires, and will handle transmission and reception of data packets. This function will also route any packets that have been received to the appropriate application protocol-level function to handle it.

The `StackApplications` function will call loaded application modules. For example, if an application is using an HTTP2 server, `StackApplications` will automatically call the `HTTPServer` function to process any HTTP2 tasks that are queued.

Most sub-tasks within `StackTask` and `StackApplications` are implemented as state-machine controlled cooperative multitasking functions. Since these sub-tasks consists of multiple steps (which may occur at varying times) this call-back system ensures that no single task will monopolize control of the processor.

Within this main loop, you may also want to poll for any I/O changes in your application and call any application specific tasks that you've implemented. To avoid causing buffer overflows in your hardware or protocol timing violations you should try to implement your own application tasks in callback functions with timing-based triggers. A method to do this is described in the next topic.

You must make one call to `StackTask` for each call of `StackApplications` but you aren't required to call these functions with any specific frequency. Calling `StackTask` too infrequently could limit your throughput, though, as each call of `StackTask` can retrieve one packet (at most) from the packet buffer. Similarly, application tasks that are time-dependant (like an ICMP ping response) may produce undesirable results if `StackApplications` is not called frequently enough.

The amount of time that the main loop takes to complete one iteration depends on several factors. If data is ready to be transmitted, or if a packet of received data was received, the `StackTask` function will take more time than it would otherwise. Each additional protocol included in your application will cause the main loop to take additional time as well, with the amount of time for each varying from the length of the shortest state machine state in the task to the longest.

Once your application is complete, you can set up a test case to determine the min/average/approximate maximum time that your loop will take to run. You can set your code up to use an internal timer to measure the duration of each iteration of the main loop, or you can set the code up to trigger an output pin each time the main loop completes, and use an oscilloscope to capture the network execution time. You can then provide application inputs or additional network traffic with a PC program (or other PICs) to simulate real-world operating conditions.

1.7.2.4 Cooperative Multitasking

Describes how to implement a non-blocking stack task.

Description

If you implement the TCP/IP stack using a cooperative multitasking approach, you must make periodic calls to task functions to transmit/receive packets and to maintain protocol functionality. To prevent conflicts with the stack, you should write your own custom tasks in a way that will allow them to give up the processor if it's not needed. If you create a protocol or application task with multiple steps, it may be beneficial to divide them up between states. You can then use a global or static variable to track your state, and call that task function periodically to move through the state machine.

The following example contains a sample application for transferring data from a machine of some type to an external target. It includes a task function called `ApplicationTask` that has states to wait for button inputs, update the display, and transfer data from the machine. The functions in the example are used to represent other actions: `ButtonPressDetected` represents the code needed to check for an input from the user, `LCDDisplay` represents the code needed to update a display on the machine, `SampleData` gets data from the machine, `DataBufferIsFull` indicates that the buffer used to hold data samples needs to be sent, and `TransferData` is a function that writes the data to an open TCP or UDP socket.

In between each of these states, the `ApplicationTask` function returns to the main loop, and the `StackTask` and `StackApplications` functions are called. This flow will allow the `StackApplications` function to maintain any module tasks. The `StackTask` function will periodically transmit the data from the socket buffers to its destination, which will prevent the transmit buffers from overflowing.

```

unsigned char gAppState;    // State tracking variable

int main (void)
{
    // Pseudo-initialization function
    InitializeCode();

    // Setup application state
    gAppState = STATE_DISPLAY_MENU;

    // Main Loop
    while (1)
    {
        StackTask();
        StackApplications();
        ApplicationTask();
    }
}

void ApplicationTask (void)
{
    switch (gAppState)
    {
        case STATE_DISPLAY_MENU:
            LCDDisplay (stringMainMenu);
            gAppState = STATE_MAIN_MENU;
            break;
        case STATE_MAIN_MENU:
            if (ButtonPressDetected (BUTTON_1) )    // Check an input
                gAppState = STATE_MONITOR_MACHINERY;
            break;
        case STATE_MONITOR_MACHINERY:
            LCDDisplay (stringTransferringData);
            // Generate or send data
            if (DataBufferIsFull())
            {
                TransferData();
            }
            else
            {
                SampleData();
            }
            if (ButtonPressDetected (BACK_BUTTON) )
                gAppState = STATE_DISPLAY_MENU;
            break;
    }
}

```

Some of the states in your application may be time based. Suppose, for example, that our sample application needs to send data for 5 seconds every time an input is detected. Stack problems could occur if the application used a delay loop to wait for 5 seconds until it was time to stop, so this functionality should be implemented using the stack's built-in tick timer. When the request to send data is received, the code will get the current tick time using the `TickGet` function, add enough ticks to make up 5 seconds, save it in a static variable called `tickCounter`, and then switch to a transmit state. Every time the `ApplicationTask` function gets called, it will enter this state in the state machine, call `TickGet` again, and then compare it to the value stored in that static variable. If the current time is later than the initial time plus the delay, the code will restore the display and re-enter the main menu state.

```

void ApplicationTask (void)
{
    static DWORD tickCounter;
    switch (gAppState)
    {
        case STATE_DISPLAY_MENU:
            LCDDisplay (stringMainMenu);

```

```
        gAppState = STATE_MAIN_MENU;
        break;
    case STATE_MAIN_MENU:
        if (ButtonPressDetected (BUTTON_1) )    // Check an input
            gAppState = STATE_MONITOR_MACHINERY;
        break;
    case STATE_MONITOR_MACHINERY:
        LCDDisplay (stringTransferringData);
        // Save the current time, and add 5 seconds to it
        tickCounter = TickGet() + (5 * TICK_SECOND);
        gAppState = STATE_CONTINUE_MONITORING;
        break;
    case STATE_CONTINUE_MONITORING:
        if ((long)(TickGet() - tickCounter) > 0)
            gAppState = STATE_DISPLAY_MENU;
        else
        {
            // Generate or send data
            if (DataBufferIsFull())
            {
                TransferData();
            }
            else
            {
                SampleData();
            }
        }
        break;
    }
}
```

There are three tick timing macros declared to help with delays: `TICK_SECOND` defines the number of ticks in a second, `TICK_MINUTE` defines the number of ticks in a minute, and `TICK_HOUR` defines the number of ticks in an hour. By using the tick timer to implement delays, you can ensure that your code won't block critical functions for too long.

1.8 Configuring the Stack

Describes how to configure the Microchip TCP/IP Stack.

Description

There is a wide range of configuration options available for Microchip's TCP/IP Stack. This topic will discuss the functionality of these options, and how to implement them.

1.8.1 Hardware Configuration

Describes how to configure the hardware options for your project.

Description

Most hardware configuration is performed by commenting, uncommenting, or defining a series of macros in the one of the variants of the header file `system_config.h` and `system.h`. You can see sample versions of how to set these options in the copies of `system_config.h` and `system.h` that are included with the stack's demo projects.

1.8.1.1 External Storage

Describes how to configure external storage options to hold web pages or structures in your application.

Description

There are several features in the TCP/IP stack that use external storage to maintain structures or web pages. Support for a few storage devices is included with the stack; the support files can be used as a template to write drivers for other devices as well. The `system_config.h` and `system.h` pin definitions are roughly equivalent for each storage device, except for the first word of the macro, which indicates which type of storage device it applies to (e.g. `EEPROM_CS_IO` vs `SPIFLASH_CS_IO`). There are three different storage media.

EEPROM

A EEPROM can be used to store MPFS2 web page images and custom application structures. To indicate to the stack that it should use a EEPROM to store MPFS2 images, define the macro `MPFS_USE_EEPROM` in the `tcpip_config.h` header file. By default, the stack includes a driver for Microchip's 25LC256 EEPROM family (to use the 1 Mbit EEPROM, you must also define the macro `USE_EEPROM_25LC1024` in `tcpip_config.h`). The macros to control communication with the EEPROM will be prepended with the string `EEPROM_` in this case. To enable communication, define `EEPROM_CS_TRIS` and include the files `spi_eeprom.c` in your application. These files may require some changes to support additional EEPROM devices.

Serial Flash

Storage for MPFS images and custom structures is also available on serial flash devices (tested with SST 25VF016B and Spansion 25FL040A). To indicate that the stack should use serial flash to store web pages, define `MPFS_USE_SPI_FLASH` in `tcpip_config.h`. The communication macros will be prepended with the string `SPIFLASH_` in this case. To enable communication functionality, define `SPIFLASH_CS_TRIS` and include the files `spi_flash.c` and `spi_flash.h` in your application. These files may require some changes to support additional flash devices. There are several macros included within "spi_flash.h" that must also be defined, including macros to define the sector and page sizes, and macros to describe whether the SST or Spansion flash device is being used.

SRAM

A serial RAM can be used to store FIFO blocks and TCP Control Blocks for sockets (tested with AMT Semiconductor's N256S0830HDA). The macros will be prepended with the string `SPIRAM_` in this case. To use this functionality, define

EEPROM_CS_TRIS and include the files “spi_ram.c” and “spi_ram.h” in your application. These files may require some changes to support additional RAM devices.

Macro	Purpose	Sample Value
xxxxxx_CS_IO	Defines the LAT (or PORT, where applicable) register bit that corresponds to the chip select pin. Defining this macro will indicate that the stack should use the specified type of external storage.	LATDbits.LATD12
xxxxxx_CS_TRIS	Defines the TRIS bit that corresponds to the chip select pin on the device.	TRISDbits.TRISD12
xxxxxx_SCK_TRIS	Defines the TRIS bit that corresponds to the clock pin of the SPI module connected to the device.	TRISGbits.TRISG6
xxxxxx_SDI_TRIS	Defines the TRIS bit that corresponds to the data-in pin of the SPI module connected to the device.	TRISGbits.TRISG7
xxxxxx_SDO_TRIS	Defines the TRIS bit that corresponds to the data-out pin of the SPI module connected to the device.	TRISGbits.TRISG8
xxxxxx_SPI_IF	Points to the interrupt flag for the SPI module connected to the device.	IFS2bits.SPI2IF
xxxxxx_SSPBUF	Points to the SPI buffer register for the SPI module connected to the device.	SPI2BUF
xxxxxx_SPICON1	Points to the SPI control register for the SPI module connected to the device.	SPI2CON1
xxxxxx_SPICON1bits	Provides bitwise access to the SPI control register for the SPI module connected to the device. The ____bits registers are typically defined in the processor's header files.	SPI2CON1bits
xxxxxx_SPICON2	Points to the second SPI control register for the SPI module connected to the device. If your device doesn't have an SPICON2 register (e.g. PIC32) just omit this definition.	SPI2CON2
xxxxxx_SPISTAT	Points to the SPI status register for the SPI module connected to the device.	SPI2STAT
xxxxxx_SPISTATbits	Provides bitwise access to the SPI status register for the SPI module connected to the device.	SPI2STATbits
xxxxxx_SPIBRG	Points to the SPI Baud Rate Generator register for the SPI module connected to the device. If your device doesn't have a BRG-based SPI module, just omit this definition.	SPI2BRG

1.8.1.2 PIC32MX7XX Config

Describes how to configure your project to use a PIC32MX795 family part.

Description

To use the PIC32MX795 in your project, include the files `ETHPIC32IntMac.c` and `ETHPIC32ExtPhy.c` in your project. You'll also have to add a specific PHY implementation file (by default `ETHPIC32ExtPhyDP83848.c` is provided) depending on your actual external PHY selection.

Update the following definitions in `system_config.h` and `system.h`:

Macro	Purpose	Sample Value
PHY_RMII	Define this macro if the external PHY runs in RMII mode. Comment it out if you're using an MII PHY.	-
PHY_CONFIG_ALTERNATE	Define this symbol if the PIC32MX7XX uses the alternate configuration pins to connect to the PHY. Comment it out for the default configuration pins.	-
PHY_ADDRESS	Update with the MIIM address of the external PHY you are using (the address on which the PHY responds to MIIM transactions. See the PHY datasheet).	0x1

Update the following definitions in `tcpip_config.h`:

Macro	Purpose	Sample Value
ETH_CFG_LINK	Set to 0 to use the default connection characteristics (depends on the selected PHY). Set to 1 to configure the Ethernet link to the following specific parameters. Auto-negotiation will always be enabled if supported by the PHY.	0
ETH_CFG_AUTO	Set to 1 to use auto negotiation. Strongly recommended.	1
ETH_CFG_10	Use/advertise 10 Mbps capability.	1
ETH_CFG_100	Use/advertise 100 Mbps capability.	1
ETH_CFG_HDUPLEX	Use/advertise half duplex capability.	1
ETH_CFG_FDUPLEX	Use/advertise full duplex capability.	1
ETH_CFG_AUTO_MDIX	Use/advertise auto MDIX capability (effective only when ETH_CFG_AUTO is enabled).	1
ETH_CFG_SWAP_MDIX	Use swapped MDIX if defined. Otherwise, use normal MDIX.	1

1.8.1.3 PIC18F87J11 Config

Describes how to configure your project to use a PIC18F87J11.

Description

The 18F87J11 can be used in your application by selecting it as the processor in MPLABX, ensuring that the ENC_CS_TRIS macro is commented out, and including the files "ETH87J11.c" and "ETH87J11.h." There are no additional macros to define for the 87J11; since it uses its own internal MAC and PHY for communication all of the register names and bit names are fixed.

1.8.2 Address

Describes how to set up addressing for your device.

Description

A TCP/IP application will need to have both a Media Access Control (MAC) address and an Internet Protocol (IP) address. There are multiple methods for obtaining or setting these addresses.

1.8.2.1 MAC Address

Describes how to configure the MAC address on your device.

Description

The 6-byte MAC address provides addressing for the Media Access Control protocol layer of the TCP/IP stack. MAC addresses are permanent addresses tied to hardware. Blocks of MAC addresses are sold to organizations and individuals by the IEEE; if you aren't using a Microchip device with a built-in MAC address, you will need to purchase one of these blocks to assign MAC addresses to your products.

The MAC address is defined in the firmware configuration header "tcpip_config.h." There are six macros that must be defined in this file to set the MAC address. They are:

Macro	Sample Value
MY_DEFAULT_MAC_BYTE1	(0x00)
MY_DEFAULT_MAC_BYTE2	(0x04)

MY_DEFAULT_MAC_BYTE3	(0xA3)
MY_DEFAULT_MAC_BYTE4	(0x00)
MY_DEFAULT_MAC_BYTE5	(0x00)
MY_DEFAULT_MAC_BYTE6	(0x00)

Each of these macros represents a byte of the MAC address (note that 00:04:A3:xx:xx:xx is the block of MAC addresses reserved for Microchip products). Once you obtain your block of addresses, you will need to specify a unique address for every device you produce. The "TCP/IP Demo App" demonstration project describes a method for using Microchip's MPLAB PM3 programmer to serially program a range of MAC addresses into multiple parts without recompiling your project.

The ENCX24J600, MRF24WB0M, MRF24WG0M and PIC32MX7XX/6XX feature a pre-programmed MAC address (from Microchip's address block). If you are using either of these part families in your project, you can define your MAC address as "00:04:A3:00:00:00" and the stack will automatically use the part's pre-programmed address for your application.

Microchip also provides a [family of EEPROMs](#) that include a unique, pre-programmed EUI-48 (MAC) or EUI-64 address. When using one of these devices, you can write your AppConfig initialization code so it will obtain the device's MAC address from one of these EEPROMs instead of the default MAC address macros.

1.8.2.2 IP Address

Describes how to configure your application's IP address.

Description

The IP address is used to address nodes on an Internet Protocol network. You will need to configure your application with an IP address, or enable a method to obtain one. You may also want to define a few other parameters that describe how your device will try to fit into its network, by default.

The macros that you will need to define include:

Macro	Property	Sample Value
MY_DEFAULT_IP_ADDR_BYTE1	Default IP address byte 1	192ul
MY_DEFAULT_IP_ADDR_BYTE2	Default IP address byte 2	168ul
MY_DEFAULT_IP_ADDR_BYTE3	Default IP address byte 3	1ul
MY_DEFAULT_IP_ADDR_BYTE4	Default IP address byte 4	100ul
MY_DEFAULT_MASK_BYTE1	Default subnet mask byte 1	255ul
MY_DEFAULT_MASK_BYTE2	Default subnet mask byte 2	255ul
MY_DEFAULT_MASK_BYTE3	Default subnet mask byte 3	255ul
MY_DEFAULT_MASK_BYTE4	Default subnet mask byte 4	0ul
MY_DEFAULT_GATE_BYTE1	Default gateway byte 1	192ul
MY_DEFAULT_GATE_BYTE2	Default gateway byte 2	168ul
MY_DEFAULT_GATE_BYTE3	Default gateway byte 3	1ul
MY_DEFAULT_GATE_BYTE4	Default gateway byte 4	1ul

The subnet address is a bit mask that defines the scope of the network. If your IP address is 192.168.5.100, and you specify a subnet mask of 255.255.255.0, the stack will assume that addresses in the range 192.168.5.x are on the same subnet that you are, and that packets sent to any of those addresses won't have to be routed anywhere else.

The default gateway is the IP address of the node on the network that your application will send packets to if it doesn't know how to route them to the address it wants to send them to. If your application is on the 192.268.5.x subnet, if it wants to send a packet to 198.175.253.160 and it doesn't know exactly how to get there, it will send it to the default gateway.

Note that if you write your own code instead of starting with a demo application, you will need to populate your AppConfig structure with these values. Also note that these are only default values. Other protocols (or your application itself) may modify any of the APP_CONFIG fields that represent these parameters.

There are three methods that you can use to set or obtain an IP address: static, DHCP, or AutoIP.

Static IP Addressing

Using a static address will allow you to specify a set IP address. This can either be done at compile time, by setting the default IP address to the value you'd like to use and using the demo code (which populated your AppConfig structure automatically), or during run-time, by programming your application to set the IP address in your AppConfig structure based on some input. If you'd like to include the code for DHCP and AutoIP address acquisition in your project but still use static addressing, you can call the DHCP and AutoIP functions that disable those modules to prevent them from overwriting your IP address. Use of static addresses will usually only work if the server is configured to support that address.

DHCP

The DHCP client module will allow your application to dynamically obtain an IP address from a DHCP server on the same network. Doing this will reset the IP address, subnet mask, gateway address, and some other configuration parameters in your AppConfig structure. To use DHCP, include the files `dhcp.c`, `dhcps.c`, and `dhcp.h` in your project, and add or uncomment the definition `#define STACK_USE_DHCP_CLIENT` to `tcpip_config.h`. The TCP/IP stack also includes a simple DHCP server that can supply an IP address to one DHCP client. To enable this functionality, add the macro `#define STACK_USE_DHCP_SERVER` to `tcpip_config.h`.

AutoIP

The AutoIP module will allow your application to choose an IP address and claim it for itself. These addresses are link-local, meaning they cannot be routed, and will only work on your local link. This functionality is based on the specification for allocating dynamic link-local addresses, but is modified to take the form used by Microsoft's APIPA link-local address allocation scheme. To enable this feature, include the files `auto_ip.c` and `auto_ip.h` and add the macro `#define STACK_USE_AUTO_IP` to `tcpip_config.h`.

IP Address Module Interaction

It is possible to configure a default static address and enable DHCP and AutoIP at the same time. If you don't disable one or the other, the AutoIP module will immediately choose an address in the specified address range and begin attempting to claim it. DHCP will also begin sending messages to attempt to lease a DHCP IP address from a DHCP server. In most cases the DHCP module will complete all of its transactions before AutoIP finishes claiming its address. In this case, the DHCP address will be copied to the AppConfig structure and the AutoIP module will stop trying to claim its address. Since a routable DHCP address is always preferred to a link-local AutoIP address, the stack will also immediately start using a DHCP address if it becomes available, even if an AutoIP address was already in use (i.e. if you enable DHCP after AutoIP has already claimed an address). This may cause existing open sockets to lose communication; they should be re-opened with the new address. In this situation, you can also use a static address if you disable DHCP and AutoIP and set the static address in the AppConfig structure.

If AutoIP is used in conjunction with the DHCP Server module, the AutoIP module will generate an address in the 169.254.x.x subnet and then serve another address in the same subnet to the DHCP client connected to the board.

1.8.3 Protocol Configuration

Describes how to set up protocols for your application.

Description

There are a few steps that you must take to include each protocol in your application. Most of this configuration is performed by setting options in one of the variants of the `tcpip_config.h` header file. Nearly all protocols will require you to enable them by defining one or more macros in `tcpip_config.h`. You will also need to include the files needed by your protocols in your project. Some protocols will require you to define sockets for them to use in `tcpip_config.h`, and allocate memory to them.

The TCP/IP Configuration Wizard, included with the stack, will allow you to select the features that you want while handling most complex firmware configuration automatically. Because of this, it is the easiest (and safest) way to set up your application protocols.

The Module APIs topic has a description of each of the modules.

1.8.3.1 Protocol Macros and Files

Describes macros and files that must be included to use protocols.

Description

You will need to define some macros in `tcpip_config.h` and include some files in your project to enable each protocol. These include:

Module	Macro	Function	Required Files
ICMP	STACK_USE_ICMP_SERVER	Provides the ability to query and respond to pings.	<code>icmp.c</code> , <code>icmp.h</code>
ICMP	STACK_USE_ICMP_CLIENT	Provides the ability to transmit pings.	<code>icmp.c</code> , <code>icmp.h</code>
HTTP2	STACK_USE_HTTP2_SERVER	Provides HTTP server functionality with dynamic variables, POST, Cookies, Authentication, and other features	<code>http2.c</code> , <code>http2.h</code> , <code>tcp.c</code> , <code>tcp.h</code> , <code>custom_http_app.c</code> and <code>http_print.h</code> (see HTTP2 section for information on these files)
SSL	STACK_USE_SSL_SERVER	Provides support for SSL server sockets.	<code>ssl.c</code> , <code>ssl.h</code> , <code>arcfour_sw.c</code> , <code>arcfour_sw.h</code> , <code>bitint.c</code> , <code>bigint.h</code> , <code>random.c</code> , <code>random.h</code> , <code>rsa_sw.c</code> , <code>rsa_sw.h</code>
SSL	STACK_USE_SSL_CLIENT	Provides support for SSL client sockets.	<code>ssl.c</code> , <code>ssl.h</code> , <code>ssl_client_size.h</code> , <code>arcfour_sw.c</code> , <code>arcfour_sw.h</code> , <code>bigint.c</code> , <code>bigint.h</code> , <code>random.c</code> , <code>random.h</code> , <code>rsa_sw.c</code> , <code>rsa_sw.h</code>
FTP	STACK_USE_FTP_SERVER	Provides ability to remotely upload MPFS2 images to HTTP2 servers via FTP	<code>ftp.c</code> , <code>ftp.h</code> , <code>tcp.c</code> , <code>tcp.h</code>
SMTP	STACK_USE_SMTP_CLIENT	Provides the ability to send email	<code>smtp.c</code> , <code>smtp.h</code> , <code>tcp.c</code> , <code>tcp.h</code> , <code>helpers.c</code> , <code>helpers.h</code>
TFTP	STACK_USE_TFTP_CLIENT	Provides unreliable file upload/download services	<code>tftp.c</code> , <code>tftp.h</code> , <code>tcp.c</code> , <code>tcp.h</code>
Telnet	STACK_USE_TELNET_SERVER	Provides telnet services.	<code>telnet.c</code> , <code>telnet.h</code> , <code>tcp.c</code> , <code>tcp.h</code>
Announce	STACK_USE_ANNOUNCE	Provides device hostname and IP address discovery on a local Ethernet subnet	<code>announce.c</code> , <code>announce.h</code>
DNS	STACK_USE_DNS	Provides the ability to resolve hostnames to IP addresses	<code>dns.c</code> , <code>dns.h</code> , <code>udp.c</code> , <code>udp.h</code>
NBNS	STACK_USE_NBNS	Provides the ability to resolve hostnames to IP addresses on the same subnet.	<code>nbns.c</code> , <code>nbns.h</code> , <code>udp.c</code> , <code>udp.h</code>
SNTP	STACK_USE_SNTP_CLIENT	Provides the ability to get the date/time from the internet	<code>sntp.c</code> , <code>sntp.h</code> , <code>udp.c</code> , <code>udp.h</code>
Dynamic DNS	STACK_USE_DYNAMICDNS_CLIENT	Provides the ability to resolve hostnames to IP addresses that change frequently.	<code>dyndns.c</code> , <code>dyndns.h</code> , <code>tcp.c</code> , <code>tcp.h</code>

MPFS2	STACK_USE_MPFS2	Provides MPFS2 services for custom applications. This functionality will be enabled/required automatically by stack-based protocols that require MPFS2.	mpfs2.c, mpfs2.h
TCP	STACK_USE_TCP	Provides TCP transport layer services for custom protocols. This functionality is automatically enabled/required by stack-based protocols that require TCP sockets.	tcp.c, tcp.h
UDP	STACK_USE_UDP	Provides UDP transport layer services for custom protocols. This functionality is automatically enabled/required by stack-based protocols that require UDP sockets.	udp.c, udp.h

1.8.3.2 Additional Features

Describes additional firmware demos and functionality.

Description

The TCP/IP stack includes some additional functionality that can be enabled in `tcipip_config.h`.

Feature	Macro	Description	Required Files
UART Demo	STACK_USE_UART	Application demo using UART for IP address display and stack configuration.	uart.c,uart.h
UART-to-TCP Bridge	STACK_USE_UART2TCP_BRIDGE	UART to TCP Bridge application example	uart_to_tcp_bridge.c, uart_totcp_brigde.h
IP Gleaning	STACK_USE_IP_GLEANING	Allows assignment of an IP address via reception of an ICMP packet with a valid IP during configuration mode	-
Reboot Server	STACK_USE_REBOOT_SERVER	Allows the PIC to be reset remotely (useful for bootloaders).	reboot.c,reboot.h

UDP Performance Test	STACK_USE_UDP_PERFORMANCE_TEST	UDP performance test. Monitor a local area network for UDP packets with a packet sniffer. This test will transmit 1024 packets. Use the timestamps of the first and last packets to calculate throughput.	udp_performance_test.c, udp_performance_test.h
TCP Performance Test	STACK_USE_TCP_PERFORMANCE_TEST	TCP performance test. Connect a demo board to a PC via UART, execute the code, and monitor the throughput on the PC terminal.	tcp_performance_test.c, tcp_performance_test.h
Berkeley API	STACK_USE_BERKELEY_API	Provides a Berkeley Sockets API abstraction layer.	berkeley_api.c, berkeley_api.h

1.8.3.3 Sockets

Describes how to set up sockets for your application.

Description

Most of your application protocols will require you to allocate memory for each connection (socket) that you have open. Like the other firmware configuration options, this is controlled by the definition of macros in `tcip_config.h`. For TCP sockets, you will have to specify four initialization parameters for each socket, including the purpose of that socket, the type of memory the socket should be stored in, the size of the transmit FIFO, and the size of the receive FIFO. The stack will then initialize the sockets with this information, and create a TCP Control Block (TCB) for each to control its operations. This topic will outline the socket configuration functionality in the sample version of `tcip_config.h` that is included with the TCP/IP Demo App project.

1.8.3.3.1 Memory Allocation

Describes memory allocation macros.

Description

```
#define TCP_ETH_RAM_SIZE (3900ul)
```

```
#define TCP_PIC_RAM_SIZE (0ul)
#define TCP_SPI_RAM_SIZE (0ul)
#define TCP_SPI_RAM_BASE_ADDRESS (0ul)
```

The first four macros in the socket section are used to describe the total amount of memory used to contain sockets. When data is sent from a TCP socket, it will first be copied into the socket's transmit FIFO, and then to the MAC/PHY transmit buffer. Similarly, received data will be read from the MAC/PHY chip into the receive FIFO. These FIFOs, as well as the TCB, can be stored in 3 places.

`TCP_ETH_RAM_SIZE` is used to define the RAM available for sockets on the actual TCP/IP MAC/PHY chip. This will not be the same as the total RAM on the chip; some memory must be reserved for packets being transmitted and received. By default ~1518 bytes (the maximum single-packet transmission size) will be reserved for TX packets on Microchip parts. The amount reserved for the receive packet buffer will equal the amount remaining after allocating the memory for the TX buffer and the memory for the sockets. You may receive a compile-time warning if the RX buffer is unreasonably small.

`TCP_PIC_RAM_SIZE` is used to define the RAM available for sockets on the PIC microcontroller that's driving your application.

`TCP_SPI_RAM_SIZE` defines the RAM available for sockets on an external SPI RAM (see External Storage). You can specify the base address in this RAM chip to use with the `TCP_SPI_RAM_BASE_ADDRESS` macro.

1.8.3.3.2 Socket Types

Describes socket types.

Description

When creating an initialization list for your sockets, you will have to specify a socket type. This parameter will define which protocol can use the socket. You can create and delete socket types as you require. In the sample version of `tcpip_config.h`, the following types are defined:

```
#define TCP_SOCKET_TYPES
#define TCP_PURPOSE_GENERIC_TCP_CLIENT 0
#define TCP_PURPOSE_GENERIC_TCP_SERVER 1
#define TCP_PURPOSE_TELNET 2
#define TCP_PURPOSE_FTP_COMMAND 3
#define TCP_PURPOSE_FTP_DATA 4
#define TCP_PURPOSE_TCP_PERFORMANCE_TX 5
#define TCP_PURPOSE_TCP_PERFORMANCE_RX 6
#define TCP_PURPOSE_UART_2_TCP_BRIDGE 7
#define TCP_PURPOSE_HTTP_SERVER 8
#define TCP_PURPOSE_DEFAULT 9
#define TCP_PURPOSE_BERKELEY_SERVER 10
#define TCP_PURPOSE_BERKELEY_CLIENT 11
#define END_OF_TCP_SOCKET_TYPES
```

The `TCP_PURPOSE_GENERIC_TCP_CLIENT` and `TCP_PURPOSE_GENERIC_TCP_SERVER` socket types are used by the generic TCP client and server examples (see `generic_tcp_client.c` and `generic_tcp_server.c`). These files are used as an example of how to create a new, custom TCP client or server application.

If you are trying to open a Telnet connection, the stack will try to use a `TCP_PURPOSE_TELNET` socket.

The `TCP_PURPOSE_FTP_COMMAND` and `TCP_PURPOSE_FTP_DATA` socket types are used to receive FTP commands and data.

The two `TCP_PERFORMANCE_x` socket types are used solely to conduct TCP performance testing.

The `TCP_PURPOSE_UART_2_TCP_BRIDGE` socket type is used for the UART-to-TCP bridge example.

The `TCP_PURPOSE_HTTP_SERVER` socket type is used for sockets on HTTP servers that listen for web page view requests.

The `TCP_PURPOSE_DEFAULT` socket type can be used for miscellaneous applications, or for applications that only need sockets temporarily. Dynamic DNS connections and SMTP connections use default sockets, and the legacy wrapper implementation for the `TCPListen` and `TCPCConnect` functions try to open them.

The `TCP_PURPOSE_BERKELEY_SERVER` and `TCP_PURPOSE_BERKELEY_CLIENT` socket types indicate that a socket is

available for the use of the Berkeley API layer (also see BSD Sockets).

1.8.3.3.3 Initialization Structure

Describes the structure used to declare and initialize TCP sockets.

Description

In the `tcPIP_config.h` header file, you must also define an array of structures to declare and initialize any sockets that you need. The sample structure is:

```
#define TCP_CONFIGURATION ROM struct {
BYTE vSocketPurpose, BYTE vMemoryMedium, WORD wTXBufferSize, WORD wRXBufferSize }
TCPSocketInitializer[] =
{
    {TCP_PURPOSE_GENERIC_TCP_CLIENT, TCP_ETH_RAM, 125, 100},
    {TCP_PURPOSE_GENERIC_TCP_SERVER, TCP_ETH_RAM, 20, 20},
    {TCP_PURPOSE_TELNET, TCP_ETH_RAM, 200, 150},
    //{TCP_PURPOSE_TELNET, TCP_ETH_RAM, 200, 150},
    //{TCP_PURPOSE_TELNET, TCP_ETH_RAM, 200, 150},
    //{TCP_PURPOSE_FTP_COMMAND, TCP_ETH_RAM, 100, 40},
    //{TCP_PURPOSE_FTP_DATA, TCP_ETH_RAM, 0, 128},
    {TCP_PURPOSE_TCP_PERFORMANCE_TX, TCP_ETH_RAM, 200, 1},
    //{TCP_PURPOSE_TCP_PERFORMANCE_RX, TCP_ETH_RAM, 40, 1500},
    {TCP_PURPOSE_UART_2_TCP_BRIDGE, TCP_ETH_RAM, 256, 256},
    {TCP_PURPOSE_HTTP_SERVER, TCP_ETH_RAM, 200, 200},
    {TCP_PURPOSE_HTTP_SERVER, TCP_ETH_RAM, 200, 200},
    {TCP_PURPOSE_DEFAULT, TCP_ETH_RAM, 200, 200},
    {TCP_PURPOSE_BERKELEY_SERVER, TCP_ETH_RAM, 25, 20},
    //{TCP_PURPOSE_BERKELEY_SERVER, TCP_ETH_RAM, 25, 20},
    //{TCP_PURPOSE_BERKELEY_SERVER, TCP_ETH_RAM, 25, 20},
    //{TCP_PURPOSE_BERKELEY_CLIENT, TCP_ETH_RAM, 125, 100},
};
#define END OF TCP CONFIGURATION
```

As you can see from the structure parameters, the four parameters you'll need to include in each of your socket declarations are:

- Socket purpose/type
- RAM storage location
- TX FIFO buffer size
- RX FIFO buffer size

Several example socket declarations are listed. The socket purpose for each corresponds to one of the socket types. The RAM storage for each socket example sets the location to `TCP_ETH_RAM` (the MAC/PHY chip RAM). Other options are `TCP_PIC_RAM` (the PIC's own RAM) and `TCP_SPI_RAM` (an external SPI RAM device). Finally, the TX and RX FIFOs are declared. Each RX buffer must contain at least one byte, to handle the SYN and FIN messages required by TCP. Each socket you declare will require up to 48 bytes of PIC RAM, and 40 + (TX FIFO size) + (RX FIFO size) bytes of RAM on the storage medium that you select.

1.8.3.3.4 UDP Sockets

Describes how UDP sockets are defined.

Description

UDP sockets are somewhat easier to declare than TCP sockets. Since UDP transmissions don't have to be processed in a particular order and responses aren't required by the sender, you don't have to declare separate buffers for these sockets. There are two options to define when using UDP:

```
#define MAX_UDP_SOCKETS      (10u)
//#define UDP_USE_TX_CHECKSUM
```

The `MAX_UDP_SOCKETS` definition defines the size of an array of `UDP_SOCKET_INFO` structures. These structures contain two sixteen-bit identifiers for the remote node's and local node's UDP port numbers, and a 10-byte structure used to hold the

remote node's MAC address and IP address (these structures use the packed attribute, so the actual size of the `UDP_SOCKET_INFO` structure may vary slightly depending on the PIC architecture you use).

The `UDP_USE_TX_CHECKSUM` definition will cause the stack to generate checksums for transmitted data, and include them with transmitted packets. This can provide some data integrity verification, but it will also decrease TX performance by nearly 50% unless the ENCX24J600 is used (the ENCX24J600 chips include hardware checksum calculators).

1.8.3.3.5 BSD Sockets

Describes the allocation of BSD sockets.

Description

The Berkeley API socket configuration option will require Berkeley sockets. Each one of these internally uses one TCP or UDP socket, defined by the `TCPsocketInitializer[]` array and the `MAX_UDP_SOCKETS` definition. Because of this, the number of Berkeley sockets you declare must be less than or equal to the sum of the number of UDP sockets you declare and the number of TCP Berkeley-type sockets you declare. The `tcpip_config.h` macro to define the number of Berkeley sockets is:

```
#define BSD_SOCKET_COUNT      (5u)
```


1.9 Library Interface

1.9.1 Stack API

The Microchip TCP/IP Stack is implemented as a suite of modules. Each module exists on its own layer in the TCP/IP layer model, and has its own set of APIs. These APIs are described in this section

1.9.1.1 Announce

Provides a UDP MAC address announcement feature.

Description

This module will facilitate device discovery on DHCP enabled networks by broadcasting a UDP message on port 30303 whenever the local IP address changes. You can change the port used by the announce module by changing the following macro definition in `announce.c`.



```
#define ANNOUNCE_PORT    30303
```

The Announce protocol is designed to be used with the TCP/IP Discoverer PC program.

1.9.1.1.1 Announce Stack Members

Functions and variables intended to be accessed only by the stack.

Functions

	Name	Description
	AnnounceIP	This is function AnnounceIP.
	DiscoveryTask	This is function DiscoveryTask.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.1.1.1 AnnounceIP Function

File

announce.h

Syntax

```
void AnnounceIP( ) ;
```

Description

This is function AnnounceIP.

1.9.1.1.1.2 DiscoveryTask Function

File

announce.h

Syntax

```
void DiscoveryTask();
```

Description

This is function DiscoveryTask.

1.9.1.2 ARCFOUR

Implements the ARCFOUR bulk encryption cipher.

Description

The ARCFOUR module implements a bulk encryption cipher that is believed to be fully interoperable with the RC4 algorithm. RC4 is trademark of RSA Data Security, Inc. It is used for bulk encryption support in the SSL module.

The ARCFOUR module is covered by US Export Control law. It must be obtained separately from Microchip.

1.9.1.2.1 ARCFOUR Public Members

Functions and variables accessible by the stack application

Structures

Name	Description
ARCFOUR_CTX	Encryption Context for ARCFOUR module. The program need not access any of these values directly, but rather only store the structure and use ARCFOURInitialize to set it up.

Description

The following functions and variables are available to the stack application.

1.9.1.2.1.1 ARCFOUR_CTX Structure

File

arc4.h

Syntax

```
typedef struct {  
    uint8_t i;  
    uint8_t j;  
    uint8_t * Sbox;  
} ARCFOUR_CTX;
```

Members

Members	Description
uint8_t i;	The iterator variable
uint8_t j;	The co-iterator
uint8_t * Sbox;	A pointer to a 256 byte S-box array

Description

Encryption Context for ARCFOUR module. The program need not access any of these values directly, but rather only store the structure and use ARCFOURInitialize to set it up.

1.9.1.3 ARP

Provides Address Resolution Protocol support.

Description

The Address Resolution Protocol, or ARP, is a foundation layer of TCP/IP. It translates IP addresses to physical MAC addresses, or locates a gateway through which a machine may be located.


TCP and UDP applications will not need to access ARP directly. The TCPOpen and UDPOpen functions will handle both ARP and DNS operations transparently.

Responses to incoming ARP requests are processed automatically. Resolution of ARP requests follows a simple state machine, as indicated in the following diagram.

1.9.1.3.1 ARP Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
	ARPRegisterCallbacks	This is function ARPRegisterCallbacks.

Macros

Name	Description
ARP_REQ	Operation code indicating an ARP Request
ARP_RESP	Operation code indicating an ARP Response

Structures

	Name	Description
	arp_app_callbacks	This is record arp_app_callbacks.

Description

The following functions and variables are available to the stack application.

1.9.1.3.1.1 ARPRegisterCallbacks Function**File**

arp.h

Syntax

```
int8_t ARPRegisterCallbacks(struct arp_app_callbacks * app);
```

Description

This is function ARPRegisterCallbacks.

1.9.1.3.1.2 arp_app_callbacks Structure**File**

arp.h

Syntax

```
struct arp_app_callbacks {
    bool used;
    void (* ARPPkt_notify)(uint32_t SenderIPAddr, uint32_t TargetIPAddr, MAC_ADDR
*SenderMACAddr, MAC_ADDR *TargetMACAddr, uint8_t op_req);
};
```

Description

This is record arp_app_callbacks.

1.9.1.3.1.3 ARP_REQ Macro

File

arp.h

Syntax

```
#define ARP_REQ 0x0001u // Operation code indicating an ARP Request
```

Description

Operation code indicating an ARP Request

1.9.1.3.1.4 ARP_RESP Macro

File

arp.h

Syntax

```
#define ARP_RESP 0x0002u // Operation code indicating an ARP Response
```



Description

Operation code indicating an ARP Response

1.9.1.3.2 ARP Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
	ARPInit	This is function ARPInit.
	ARPPProcess	This is function ARPPProcess.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.3.2.1 ARPInit Function

File

arp.h

Syntax

```
void ARPInit();
```

Description

This is function ARPInit.

1.9.1.3.2.2 ARPPProcess Function

File

arp.h

Syntax

```
bool ARPPProcess();
```

Description

This is function ARPPProcess.

1.9.1.3.3 ARP Internal Members

Functions and variables internal to the ARP module

Macros

Name	Description
ARP_OPERATION_REQ	Operation code indicating an ARP Request
ARP_OPERATION_RESP	Operation code indicating an ARP Response
HW_ETHERNET	ARP Hardware type as defined by IEEE 802.3
ARP_IP	ARP IP packet type as defined by IEEE 802.3

Description

The following functions and variables are designated as internal to the ARP module.

1.9.1.3.3.1 ARP_OPERATION_REQ Macro

File

arp.h

Syntax

```
#define ARP_OPERATION_REQ 0x0001u // Operation code indicating an ARP Request
```

Description

Operation code indicating an ARP Request

1.9.1.3.3.2 ARP_OPERATION_RESP Macro

File

arp.h

Syntax

```
#define ARP_OPERATION_RESP 0x0002u // Operation code indicating an ARP Response
```

Description

Operation code indicating an ARP Response

1.9.1.3.3.3 HW_ETHERNET Macro

File

arp.h

Syntax

```
#define HW_ETHERNET (0x0001u) // ARP Hardware type as defined by IEEE 802.3
```

Description

ARP Hardware type as defined by IEEE 802.3

1.9.1.3.3.4 ARP_IP Macro

File

arp.h

Syntax

```
#define ARP_IP (0x0800u) // ARP IP packet type as defined by IEEE 802.3
```

Description

ARP IP packet type as defined by IEEE 802.3

1.9.1.4 Berkeley (BSD) Sockets

Provides a BSD socket wrapper to the Microchip TCP/IP Stack.

Description












The Berkeley Socket Distribution (BSD) APIs provide a BSD wrapper to the native Microchip TCP/IP Stack APIs. Using this interface, programmers familiar with BSD sockets can quickly develop applications using Microchip's TCP/IP Stack.

The illustration below shows a typical interaction for a TCP server or client using the Berkeley socket APIs.

1.9.1.4.1 BSD Wrapper Public Members

Functions and variables accessible by the stack application



Functions



	Name	Description
	accept	This is function accept.
	bind	This is function bind.
	closesocket	This is function closesocket.
	connect	This is function connect.
	gethostname	This is function gethostname.
	recv	This is function recv.
	recvfrom	This is function recvfrom.
	send	This is function send.
	sendto	This is function sendto.
	listen	This is function listen.
	socket	This is function socket.

Macros

Name	Description
AF_INET	Internet Address Family - UDP, TCP, etc.
INADDR_ANY	IP address for server binding.
INVALID_TCP_PORT	Invalid TCP port
IP_ADDR_ANY	IP Address for server binding
IPPROTO_IP	Indicates IP pseudo-protocol.
IPPROTO_TCP	Indicates TCP for the internet address family.
IPPROTO_UDP	Indicates UDP for the internet address family.
SOCK_DGRAM	Connectionless datagram socket. Use UDP for the internet address family.
SOCK_STREAM	Connection based byte streams. Use TCP for the internet address family.
SOCKET_CNXN_IN_PROGRESS	Socket connection state.
SOCKET_DISCONNECTED	Socket disconnected
SOCKET_ERROR	Socket error

Structures

	Name	Description
	BSDSocket	Berkeley Socket structure
	in_addr	in_addr structure

	sockaddr	generic address structure for all address families
	sockaddr_in	In the Internet address family

Types

Name	Description
SOCKADDR	generic address structure for all address families
SOCKADDR_IN	In the Internet address family
SOCKET	Socket descriptor

Description

The following functions and variables are available to the stack application.

1.9.1.4.1.1 accept Function

File

berkeley_api.h

Syntax

```
SOCKET accept(SOCKET s, struct sockaddr * addr, int * addrlen);
```

Description

This is function accept.

1.9.1.4.1.2 AF_INET Macro

File

berkeley_api.h

Syntax

```
#define AF_INET 2 // Internet Address Family - UDP, TCP, etc.
```

Description

Internet Address Family - UDP, TCP, etc.

1.9.1.4.1.3 bind Function

File

berkeley_api.h

Syntax

```
int bind(SOCKET s, const struct sockaddr * name, int namelen);
```

Description

This is function bind.

1.9.1.4.1.4 BSDSocket Structure

File

berkeley_api.h

Syntax

```
struct BSDSocket {
    int SocketType;
    BSD_SCK_STATE bsdState;
    uint16_t localPort;
    uint16_t remotePort;
    uint32_t remoteIP;
```

```
int backlog;
bool isServer;
TCP_SOCKET SocketID;
};
```

Members

Members	Description
int SocketType;	Socket type
BSD_SCK_STATE bsdState;	Socket state
uint16_t localPort;	local port
uint16_t remotePort;	remote port
uint32_t remoteIP;	remote IP
int backlog;	maximum number of client connections
bool isServer;	server/client check
TCP_SOCKET SocketID;	Socket ID

Description

Berkeley Socket structure

1.9.1.4.1.5 closesocket Function

File

berkeley_api.h

Syntax

```
int closesocket(SOCKET s);
```

Description

This is function closesocket.

1.9.1.4.1.6 connect Function

File

berkeley_api.h

Syntax

```
int connect(SOCKET s, struct sockaddr * name, int namelen);
```

Description

This is function connect.

1.9.1.4.1.7 gethostname Function

File

berkeley_api.h

Syntax

```
int gethostname(char * name, int namelen);
```

Description

This is function gethostname.

1.9.1.4.1.8 in_addr Structure

File

berkeley_api.h

Syntax

```
struct in_addr {
    union {
        struct {
            uint8_t s_b1, s_b2, s_b3, s_b4;
        } S_un_b;
        struct {
            uint16_t s_w1, s_w2;
        } S_un_w;
        uint32_t S_addr;
    } S_un;
};
```

Members

Members	Description
union { struct { uint8_t s_b1, s_b2, s_b3, s_b4; } S_un_b; struct { uint16_t s_w1, s_w2; } S_un_w; uint32_t S_addr; } S_un;	union of IP address
struct { uint8_t s_b1, s_b2, s_b3, s_b4; } S_un_b;	IP address in Byte
struct { uint16_t s_w1, s_w2; } S_un_w;	IP address in Word
uint32_t S_addr;	IP address

Description

in_addr structure

1.9.1.4.1.9 recv Function

File

berkeley_api.h

Syntax

```
int recv(SOCKET s, char * buf, int len, int flags);
```

Description

This is function recv.

1.9.1.4.1.10 INADDR_ANY Macro

File

berkeley_api.h

Syntax

```
#define INADDR_ANY 0x00000000u // IP address for server binding.
```

Description

IP address for server binding.

1.9.1.4.1.11 recvfrom Function

File

berkeley_api.h

Syntax

```
int recvfrom(SOCKET s, char * buf, int len, int flags, struct sockaddr * from, int * fromlen);
```

Description

This is function recvfrom.

1.9.1.4.1.12 INVALID_TCP_PORT Macro

File

berkeley_api.h

Syntax

```
#define INVALID_TCP_PORT (0L) // Invalid TCP port
```

Description

Invalid TCP port

1.9.1.4.1.13 send Function

File

berkeley_api.h

Syntax

```
int send(SOCKET s, const char * buf, int len, int flags);
```

Description

This is function send.

1.9.1.4.1.14 IP_ADDR_ANY Macro

File

berkeley_api.h

Syntax

```
#define IP_ADDR_ANY 0u // IP Address for server binding
```

Description

IP Address for server binding

1.9.1.4.1.15 sendto Function

File

berkeley_api.h

Syntax

```
int sendto(SOCKET s, const char * buf, int len, int flags, const struct sockaddr * to, int tolen);
```

Description

This is function sendto.

1.9.1.4.1.16 IPPROTO_IP Macro

File

berkeley_api.h

Syntax

```
#define IPPROTO_IP 0 // Indicates IP pseudo-protocol.
```

Description

Indicates IP pseudo-protocol.

1.9.1.4.1.17 IPPROTO_TCP Macro

File

berkeley_api.h

Syntax

```
#define IPPROTO_TCP 6 // Indicates TCP for the internet address family.
```

Description

Indicates TCP for the internet address family.

1.9.1.4.1.18 IPPROTO_UDP Macro

File

berkeley_api.h

Syntax

```
#define IPPROTO_UDP 17 // Indicates UDP for the internet address family.
```

Description

Indicates UDP for the internet address family.

1.9.1.4.1.19 listen Function

File

berkeley_api.h

Syntax

```
int listen(SOCKET s, int backlog);
```

Description

This is function listen.

1.9.1.4.1.20 SOCK_DGRAM Macro

File

berkeley_api.h

Syntax

```
#define SOCK_DGRAM 110 // Connectionless datagram socket. Use UDP for the internet address family.
```

Description

Connectionless datagram socket. Use UDP for the internet address family.

1.9.1.4.1.21 SOCK_STREAM Macro

File
berkeley_api.h

Syntax
`#define SOCK_STREAM 100 // Connection based byte streams. Use TCP for the internet address family.`

Description
Connection based byte streams. Use TCP for the internet address family.

1.9.1.4.1.22 sockaddr Structure

File
berkeley_api.h

Syntax
`struct sockaddr {
 unsigned short sa_family;
 char sa_data[14];
};`

Members

Members	Description
unsigned short sa_family;	address family
char sa_data[14];	up to 14 bytes of direct address

Description
generic address structure for all address families

1.9.1.4.1.23 SOCKADDR Type

File
berkeley_api.h

Syntax
`typedef struct sockaddr SOCKADDR;`

Description
generic address structure for all address families

1.9.1.4.1.24 sockaddr_in Structure

File
berkeley_api.h

Syntax
`struct sockaddr_in {
 short sin_family;
 uint16_t sin_port;
 struct in_addr sin_addr;
 char sin_zero[8];
};`

Members

Members	Description
short sin_family;	Address family; must be AF_INET.

uint16_t sin_port;	Internet Protocol (IP) port.
struct in_addr sin_addr;	IP address in network byte order.
char sin_zero[8];	Padding to make structure the same size as SOCKADDR.

Description

In the Internet address family

1.9.1.4.1.25 SOCKADDR_IN Type

File

berkeley_api.h

Syntax

```
typedef struct sockaddr_in SOCKADDR_IN;
```

Description

In the Internet address family

1.9.1.4.1.26 socket Function

File

berkeley_api.h

Syntax

```
SOCKET socket(int af, int type, int protocol);
```

Description

This is function socket.

1.9.1.4.1.27 SOCKET Type

File

berkeley_api.h

Syntax

```
typedef uint8_t SOCKET;
```

Description

Socket descriptor

1.9.1.4.1.28 SOCKET_CNXN_IN_PROGRESS Macro

File

berkeley_api.h

Syntax

```
#define SOCKET_CNXN_IN_PROGRESS (-2) // Socket connection state.
```

Description

Socket connection state.

1.9.1.4.1.29 SOCKET_DISCONNECTED Macro

File

berkeley_api.h

Syntax

```
#define SOCKET_DISCONNECTED (-3) // Socket disconnected
```

Description

Socket disconnected

1.9.1.4.1.30 SOCKET_ERROR Macro**File**

berkeley_api.h

Syntax

```
#define SOCKET_ERROR (-1) // Socket error
```


Description

Socket error

1.9.1.4.2 BSD Wrapper Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
	BerkeleySocketInit	This is function BerkeleySocketInit.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.4.2.1 BerkeleySocketInit Function**File**

berkeley_api.h

Syntax

```
void BerkeleySocketInit();
```

Description

This is function BerkeleySocketInit.

1.9.1.4.3 BSD Wrapper Internal Members

Functions and variables internal to the module

Types

Name	Description
BSD_SCK_STATE	Berkeley Socket (BSD) states

Description

The following functions and variables are designated as internal to the module.

1.9.1.4.3.1 BSD_SCK_STATE Type**File**

berkeley_api.h

Syntax

```
typedef enum BSD_SCK_STATE@1 BSD_SCK_STATE;
```

Description

Berkeley Socket (BSD) states

1.9.1.5 DNS Client

Provides Domain Name Service resolution.

Description

The Domain Name Service associates host names (such as www.microchip.com) with IP addresses (such as 10.0.54.2). The DNS Client module provides DNS resolution capabilities to the stack.

TCP applications do not need to use the DNS module. Any necessary DNS operations can be handled by the TCPOpen function. Applications built using UDP may need to use DNS when the IP address of the remote server is unknown.

DNS resolution operations follow a simple state machine, as indicated in the diagram below.

1.9.1.5.1 DNS Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
≡	DNSBeginUsage	This is function DNSBeginUsage.
≡	DNSEndUsage	This is function DNSEndUsage.

Macros

Name	Description
DNS_TYPE_A	Constant for record type in DNSResolve. Indicates an A (standard address) record.
DNS_TYPE_MX	Constant for record type in DNSResolve. Indicates an MX (mail exchanger) record.

Description

The following functions and variables are available to the stack application.

1.9.1.5.1.1 DNSBeginUsage Function

File

dns.h

Syntax

```
bool DNSBeginUsage();
```

Description

This is function DNSBeginUsage.

1.9.1.5.1.2 DNSEndUsage Function

File

dns.h

Syntax

```
bool DNSEndUsage();
```

Description

This is function DNSEndUsage.

1.9.1.5.1.3 DNS_TYPE_A Macro

File

dns.h

Syntax

```
#define DNS_TYPE_A (1u) // Constant for record type in DNSResolve. Indicates an A (standard address) record.
```

Description

Constant for record type in DNSResolve. Indicates an A (standard address) record.

1.9.1.5.1.4 DNS_TYPE_MX Macro

File

dns.h

Syntax

```
#define DNS_TYPE_MX (15u) // Constant for record type in DNSResolve. Indicates an MX (mail exchanger) record.
```


Description

Constant for record type in DNSResolve. Indicates an MX (mail exchanger) record.

1.9.1.5.2 DNS Internal Members

Functions and variables internal to the DNS module

Functions

	Name	Description
	DNSResolveROM	Non-ROM variant for C30/C32

Description

The following functions and variables are designated as internal to the DNS module.

1.9.1.5.2.1 DNSResolveROM Function

File

dns.h

Syntax

```
void DNSResolveROM(ROM uint8_t * Hostname, uint8_t Type);
```

Description

Non-ROM variant for C30/C32

1.9.1.6 Dynamic DNS Client

Updates an external IP address to a Dynamic DNS service.

Description

The Dynamic DNS Client module provides a method for updating a dynamic IP address to a public DDNS service. These services can be used to provide DNS hostname mapping to devices that behind routers, firewalls, and/or on networks that dynamically assign IP addresses.

Note that this only solves one of the two problems for communicating to devices on local subnets from the Internet. While Dynamic DNS can help to locate the device, the router or firewall it sits behind must still properly forward the incoming connection request. This generally requires port forwarding to be configured for the router behind which the device is located.

The Dynamic DNS client supports the popular interface used by DynDNS.org, No-IP.com, and DNS-O-Matic.com.

IMPORTANT: The dynamic DNS services stipulate that updates should be made no more frequently than 10 minutes, and only when the IP address has changed. Updates made more often than that are considered abusive, and may eventually cause your account to be disabled. Production devices that get rebooted frequently may need to store the last known IP in non-volatile memory. You also should not enable this module while testing the rest of your application.

1.9.1.6.1 Dynamic DNS Public Members

Functions and variables accessed by the stack application

Enumerations

Name	Description
DDNS_SERVICES	Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to ddnsServiceHosts and ddnsServicePorts in ddns.c.
DDNS_STATUS	Status message for DynDNS client. GOOD and NOCHG are OK, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

Functions

	Name	Description
≡	DDNSForceUpdate	This is function DDNSForceUpdate.
≡	DDNSGetLastIP	This is function DDNSGetLastIP.
≡	DDNSGetLastStatus	This is function DDNSGetLastStatus.
≡	DDNSSetService	This is function DDNSSetService.

Structures

Name	Description
DDNS_POINTERS	Configuration parameters for the Dynamic DNS Client

Variables

Name	Description
DDNSClient	Global DDNS Configuration parameters

Description

These functions and variables are meant to be called by your stack application.

1.9.1.6.1.1 DDNS_POINTERS Structure

Configuration parameters for the Dynamic DNS Client

File

ddns.h

Syntax

```
typedef struct {  
    union {
```

```
uint8_t * szRAM;
ROM uint8_t * szROM;
} CheckIPServer;
uint16_t CheckIPPort;
union {
    uint8_t * szRAM;
    ROM uint8_t * szROM;
} UpdateServer;
uint16_t UpdatePort;
union {
    uint8_t * szRAM;
    ROM uint8_t * szROM;
} Username;
union {
    uint8_t * szRAM;
    ROM uint8_t * szROM;
} Password;
union {
    uint8_t * szRAM;
    ROM uint8_t * szROM;
} Host;
struct {
    unsigned char CheckIPServer : 1;
    unsigned char UpdateServer : 1;
    unsigned char Username : 1;
    unsigned char Password : 1;
    unsigned char Host : 1;
} ROMPointers;
} DDNS_POINTERS;
```

Description

This structure of pointers configures the Dynamic DNS Client. Initially, all pointers will be null and the client will be disabled. Set DDNSClient.[field name].szRAM to use a string stored in RAM, or DDNSClient.[field name].szROM to use a string stored in ROM. (Where [field name] is one of the parameters below.)

If a ROM string is specified, DDNSClient.ROMPointers.[field name] must also be set to 1 to indicate that this field should be retrieved from ROM instead of RAM.

Parameters

Parameters	Description
CheckIPServer	The server used to determine the external IP address
CheckIPPort	Port on the above server to connect to
UpdateServer	The server where updates should be posted
UpdatePort	Port on the above server to connect to
Username	The user name for the dynamic DNS server
Password	The password to supply when making updates
Host	The host name you wish to update
ROMPointers	Indicates which parameters to read from ROM instead of RAM.

1.9.1.6.1.2 DDNS_SERVICES Enumeration

File

ddns.h

Syntax

```
typedef enum {
    DYNDNS_ORG = 0u,
    NO_IP_COM,
    DNSOMATIC_COM
} DDNS_SERVICES;
```

Members

Members	Description
DYNDNS_ORG = 0u	www.dyndns.org
NO_IP_COM	www.no-ip.com
DNSOMATIC_COM	www.dnsomatic.com

Description

Dynamic DNS Services. Must support the DynDNS API (Auxlang) and correspond to ddnsServiceHosts and ddnsServicePorts in ddns.c.

1.9.1.6.1.3 DDNS_STATUS Enumeration**File**

ddns.h

Syntax

```
typedef enum {
    DDNS_STATUS_GOOD = 0u,
    DDNS_STATUS_NOCHG,
    DDNS_STATUS_ABUSE,
    DDNS_STATUS_BADSYS,
    DDNS_STATUS_BADAGENT,
    DDNS_STATUS_BDAUTH,
    DDNS_STATUS_NOT_DONATOR,
    DDNS_STATUS_NOT_FQDN,
    DDNS_STATUS_NOHOST,
    DDNS_STATUS_NOT_YOURS,
    DDNS_STATUS_NUMHOST,
    DDNS_STATUS_DNSERR,
    DDNS_STATUS_911,
    DDNS_STATUS_UPDATE_ERROR,
    DDNS_STATUS_UNCHANGED,
    DDNS_STATUS_CHECKIP_ERROR,
    DDNS_STATUS_INVALID,
    DDNS_STATUS_UNKNOWN
} DDNS_STATUS;
```

Members

Members	Description
DDNS_STATUS_GOOD = 0u	Update successful, hostname is now updated
DDNS_STATUS_NOCHG	Update changed no setting and is considered abusive. Additional 'nochg' updates will cause hostname to be blocked.
DDNS_STATUS_ABUSE	The hostname specified is blocked for update abuse.
DDNS_STATUS_BADSYS	System parameter not valid. Should be dyndns, statdns or custom.
DDNS_STATUS_BADAGENT	The user agent was blocked or not sent.
DDNS_STATUS_BDAUTH	The username and password pair do not match a real user.
DDNS_STATUS_NOT_DONATOR	An option available only to credited users (such as offline URL) was specified, but the user is not a credited user. If multiple hosts were specified, only a single !donator will be returned.
DDNS_STATUS_NOT_FQDN	The hostname specified is not a fully-qualified domain name (not in the form hostname.dyndns.org or domain.com).
DDNS_STATUS_NOHOST	The hostname specified does not exist in this user account (or is not in the service specified in the system parameter).
DDNS_STATUS_NOT_YOURS	The hostname specified does not belong to this user account.
DDNS_STATUS_NUMHOST	Too many hosts specified in an update.

DDNS_STATUS_DNSERR	Unspecified DNS error encountered by the DDNS service.
DDNS_STATUS_911	There is a problem or scheduled maintenance with the DDNS service.
DDNS_STATUS_UPDATE_ERROR	Error communicating with Update service.
DDNS_STATUS_UNCHANGED	The IP Check indicated that no update was necessary.
DDNS_STATUS_CHECKIP_ERROR	Error communicating with CheckIP service.
DDNS_STATUS_INVALID	DDNS Client data is not valid.
DDNS_STATUS_UNKNOWN	DDNS client has not yet been executed with this configuration.

Description

Status message for DynDNS client. GOOD and NOCHG are OK, but ABUSE through 911 are fatal. UNCHANGED through INVALID are locally defined.

1.9.1.6.1.4 DDNSClient Variable**File**

ddns.h

Syntax

```
DDNS_POINTERS DDNSClient;
```

Description

Global DDNS Configuration parameters

1.9.1.6.1.5 DDNSForceUpdate Function**File**

ddns.h

Syntax

```
void DDNSForceUpdate();
```

Description

This is function DDNSForceUpdate.

1.9.1.6.1.6 DDNSGetLastIP Function**File**

ddns.h

Syntax

```
IP_ADDR DDNSGetLastIP();
```

Description

This is function DDNSGetLastIP.

1.9.1.6.1.7 DDNSGetLastStatus Function**File**

ddns.h

Syntax

```
DDNS_STATUS DDNSGetLastStatus();
```

Description

This is function DDNSGetLastStatus.

1.9.1.6.1.8 DDNSSetService Function

File

ddns.h

Syntax

```
void DDNSSetService(DDNS_SERVICES svc);
```



Description

This is function DDNSSetService.

1.9.1.6.2 Dynamic DNS Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
	DDNSInit	This is function DDNSInit.
	DDNSTask	

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.6.2.1 DDNSInit Function

File

ddns.h

Syntax

```
void DDNSInit();
```

Description

This is function DDNSInit.

1.9.1.6.2.2 DDNSTask Function

File

ddns.h

Syntax

```
void DDNSTask();
```

Section

Function Prototypes

1.9.1.6.3 Dynamic DNS Internal Members

Functions and variables internal to the Dynamic DNS module

Macros

Name	Description
DDNS_CHECKIP_SERVER	Default CheckIP server for determining current IP address
DDNS_DEFAULT_PORT	Default port for CheckIP server

Description

The following functions and variables are designated as internal to the Dynamic DNS module.

1.9.1.6.3.1 DDNS_CHECKIP_SERVER Macro

File

ddns.h

Syntax

```
#define DDNS_CHECKIP_SERVER (ROM uint8_t *)"checkip.dyndns.com" // Default CheckIP server  
for determining current IP address
```

Description

Default CheckIP server for determining current IP address

1.9.1.6.3.2 DDNS_DEFAULT_PORT Macro

File

ddns.h

Syntax

```
#define DDNS_DEFAULT_PORT (80u) // Default port for CheckIP server
```

Description

Default port for CheckIP server

1.9.1.7 Hashes

Calculates MD5 and SHA-1 hash sums.

Description

The Hashes module calculates MD5 and/or SHA-1 hash sums of data. Hash sums are one-way digest functions, meaning that the original message cannot be derived from the hash of the message. Collisions, while exceedingly rare, do exist. However, they are extremely difficult to create.

Hash functions are generally used for message integrity and authentication purposes. They are used extensively by encryption protocols such as SSL to verify that a message has not been tampered with during transit.


The following flow diagram demonstrates how to use this module.

To use the hash functions, first declare a HASH_SUM structure and pass a pointer to it to either MD5Initialize or SHA1Initialize. Then, call HashAddData or HashAddROMData as many times as are necessary to provide all the data to the hash. Call MD5Calculate or SHA1Calculate at any time to obtain the hash sum up to the current point. After calculation, continue adding data and repeating this process as many times as necessary.

1.9.1.7.1 Hashes Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
	HashAddROMData	Non-ROM variant for C30 / C32

Structures

Name	Description
HASH_SUM	Context storage for a hash operation

Description

The following functions and variables are available to the stack application.

1.9.1.7.1.1 HashAddROMData Function**File**

hashes.h

Syntax

```
void HashAddROMData(HASH_SUM * theSum, ROM uint8_t * data, uint16_t len);
```

Description

Non-ROM variant for C30 / C32

1.9.1.7.1.2 HASH_SUM Structure**File**

hashes.h

Syntax

```
typedef struct {
    uint32_t h0;
    uint32_t h1;
    uint32_t h2;
    uint32_t h3;
    uint32_t h4;
    uint32_t bytesSoFar;
    uint8_t partialBlock[64];
    HASH_TYPE hashType;
} HASH_SUM;
```

Members

Members	Description
uint32_t h0;	Hash state h0
uint32_t h1;	Hash state h1
uint32_t h2;	Hash state h2
uint32_t h3;	Hash state h3
uint32_t h4;	Hash state h4
uint32_t bytesSoFar;	Total number of bytes hashed so far
uint8_t partialBlock[64];	Beginning of next 64 byte block
HASH_TYPE hashType;	Type of hash being calculated

Description

Context storage for a hash operation

1.9.1.7.2 Hashes Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
≡	MD5AddROMData	Non-ROM variant for C30 / C32
≡	SHA1AddROMData	Non-ROM variant for C30 / C32

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.7.2.1 MD5AddROMData Function

File

hashes.h

Syntax

```
void MD5AddROMData(HASH_SUM * theSum, ROM uint8_t * data, uint16_t len);
```

Description

Non-ROM variant for C30 / C32

1.9.1.7.2.2 SHA1AddROMData Function

File

hashes.h

Syntax

```
void SHA1AddROMData(HASH_SUM * theSum, ROM uint8_t * data, uint16_t len);
```

Description

Non-ROM variant for C30 / C32

1.9.1.7.3 Hashes Internal Members

Functions and variables internal to the Hashes module

Enumerations

Name	Description
HASH_TYPE	Type of hash being calculated

Description

The following functions and variables are designated as internal to the Hashes module.

1.9.1.7.3.1 HASH_TYPE Enumeration

File

hashes.h

Syntax

```
typedef enum {  
    HASH_MD5 = 0u,  
    HASH_SHA1  
} HASH_TYPE;
```


Members

Members	Description
HASH_MD5 = 0u	MD5 is being calculated
HASH_SHA1	SHA-1 is being calculated

Description

Type of hash being calculated

1.9.1.8 Helpers

Provides several helper function for stack operation.

Description

This module contains several helper functions used throughout the TCP/IP Stack. Some of these duplicate functionality already implemented in the compiler's default libraries. In those cases, the compiler's version is used and the stack's version is omitted.

1.9.1.8.1 Helpers Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
≡	GenerateRandomDWORD	This is function GenerateRandomDWORD.
≡	leftRotateDWORD	Rotations are more efficient in PIC24 and PIC32
≡	ROMStringToIPAddress	Non-ROM variant for PIC24 and PIC32
≡	strnchr	This is function strnchr.

Macros

Name	Description
ultoa	This is macro ultoa.

Description

The following functions and variables are available to the stack application.

1.9.1.8.1.1 GenerateRandomDWORD Function

File

helpers.h

Syntax

```
uint32_t GenerateRandomDWORD();
```

Description

This is function GenerateRandomDWORD.

1.9.1.8.1.2 leftRotateDWORD Function

File

helpers.h

Syntax

```
uint32_t leftRotateDWORD(uint32_t val, uint8_t bits);
```

Description

Rotations are more efficient in PIC24 and PIC32

1.9.1.8.1.3 ROMStringToIPAddress Function**File**

helpers.h

Syntax

```
bool ROMStringToIPAddress(ROM uint8_t * str, IP_ADDR* IPAddress);
```

Description

Non-ROM variant for PIC24 and PIC32

1.9.1.8.1.4 strnchr Function**File**

helpers.h

Syntax

```
char * strnchr(const char * searchString, size_t count, char c);
```

Description

This is function strnchr.

1.9.1.8.1.5 ultoa Macro**File**

helpers.h

Syntax

```
#define ultoa(val,buf) ultoa((char *) (buf), (val), 10)
```

Description

This is macro ultoa.

1.9.1.9 HTTP2 Server

Provides an advanced embedded web server.

Description

The HTTP2 web server module and its associated MPFS2 file system module allow the board to act as a web server. This facilitates an easy method to view status information and control applications using any standard web browser.

Three main components are necessary to understand how the HTTP2 web server works: the web pages, the MPFS2 Utility, and the source files `custom_http_app.c` and `http_print.h`. An overview of the entire process is shown below.

Web Pages

This includes all the HTML and associated images, CSS stylesheets, and JavaScript files necessary to display the website. A sample application including all these components is located in the WebPages2 folder.

MPFS2 Utility

This program, supplied by Microchip, packages the web pages into a format that can be efficiently stored in either external non-volatile storage, or internal flash program memory. This program also indexes dynamic variables found in the web pages and updates `http_print.h` with these indices.

If external storage is being used, the MPFS2 Utility outputs a BIN file and can upload that file directly to the board. If the data is being stored in Flash program memory, the MPFS2 Utility will generate a C source file image to be included in the project.

When dynamic variables are added or removed from your application, the MPFS2 Utility will update `http_print.h`. When this happens, the project must be recompiled in the MPLABX IDE to ensure that all the new variable indices get added into the application.

custom_http_app.c

This file implements the web application. It describes the output for dynamic variables (via HTTPPrint_varname callbacks), parses data submitted through forms (in HTTPExecuteGet and HTTPExecutePost) and validates authorization credentials (in HTTPAuthenticate). The exact functionality of these callbacks is described within the demo application's web pages, and is also documented within the `custom_http_app.c` example that is distributed with the stack.

HTTPPrint.h

This file is generated automatically by the MPFS2 Utility. It indexes all the dynamic variables and provides the "glue" between the variables located in the web pages and their associated HTTPPrint_varname callback functions defined in `custom_http_app.c`. This file does not require modification by the programmer.

1.9.1.9.1 HTTP2 Features

Features available in the HTTP2 module

Description

The HTTP2 web server module has many capabilities. The following topics will introduce these features and provide examples.

1.9.1.9.1.1 HTTP2 Dynamic Variables

Displays real-time data in web pages or other outputs

Description

One of the most basic needs is to provide status information back to the user of your web application. The HTTP server provides for this using dynamic variable substitution callbacks. These commands in your HTML code will alert the server to execute a callback function at that point, which the developer creates to write data into the web page. Dynamic Variables should be considered the output of your application.

Basic Use

To create a dynamic variable, simply enclose the name of the variable inside a pair of tilde (~) characters within the web pages' HTML source code. (ex: `~myVariable~`) When you run the MPFS2 Utility to generate the web pages, it will automatically index these variables in `http_print.h`. This index will instruct your application to invoke the function `HTTPPrint_myVariable` when this string is encountered.

Here is an example of using a dynamic variable to insert the build date of your application into the web pages:

```
<div class="examplebox code">~builddate~</div>
```

The associated callback will print the value into the web page:

```
void HTTPPrint_builddate(void)
{
    TCPPutROMString(sktHTTP, (ROM void*)__DATE__);
}
```

Passing Parameters

You can also pass parameters to dynamic variables by placing numeric values inside of parenthesis after the variable name. For example, `~led(2)~` will print the value of the second LED. The numeric values are passed as WORD values to your callback function. You can pass as many parameters as you wish to these functions, and if your C code has constants defined, those will be parsed as well. (ex: `~pair(3,TRUE)~`)

The following code inserts the value of the push buttons into the web page, all using the same callback function:

```
<div class="examplebox code">btn(3)~ btn(2)~ btn(1)~ btn(0)~</div>
```

This associated callback will print the value of the requested button to the web page:

```
void HTTPPrint_btn(WORD num)
{
    // Determine which button
    switch(num)
    {
        case 0:
            num = BUTTON0_IO;
            break;
        case 1:
            num = BUTTON1_IO;
            break;
        case 2:
            num = BUTTON2_IO;
            break;
        case 3:
            num = BUTTON3_IO;
            break;
        default:
            num = 0;
    }

    // Print the output
    if(num == 1)
        TCPPutROMString(sktHTTP, "up");
    else
        TCPPutROMString(sktHTTP, "down");
}
```

Longer Outputs

The HTTP protocol operates in a fixed memory buffer for transmission, so not all data can be sent at once. Care must be taken inside of your callback function to avoid overrunning this buffer.

The HTTP2 web server verifies that at least 16 bytes are free in this buffer before invoking a callback. For short outputs (less than 16 bytes), callbacks need only to call the appropriate TCPPut function and return. For longer outputs, callback functions must check how much space is available, write up to that many bytes, then return. The callback will be invoked again when more space is free.

To manage the output state, callbacks should make use of `curHTTP.callbackPos`. This DWORD value is set to zero when a callback is first invoked. If a callback is only writing part of its output, it should set this field to a non-zero value to indicate that it should be called again when more space is available. This value will be available to the callback during the next call, which allows the function to resume output where it left off. A common use is to store the number of bytes written, or remaining to be written, in this field. Once the callback is finished writing its output, it must set `curHTTP.callbackPos` back to zero in order to indicate completion.

As an example, this code outputs the current value of the LCD display, which is 32 bytes on many Microchip development boards:

```
<div class="examplebox code">~lcdtext~</div>
```

The following callback function handles the output, and manages its state for multiple calls:

```
void HTTPPrint_lcdtext(void)
{
    WORD len;

    // Determine how many bytes we can write
    len = TCPIsPutReady(sktHTTP);

    // If just starting, set callbackPos
    if(curHTTP.callbackPos == 0)
        curHTTP.callbackPos = 32;

    // Write a byte at a time while we still can
```

```

// It may take up to 12 bytes to write a character
// (spaces and newlines are longer)
while(len > 12 && curHTTP.callbackPos)
{
    // After 16 bytes write a newline
    if(curHTTP.callbackPos == 16)
        len -= TCPPutROMString(sktHTTP, (ROM BYTE*)"<br />");

    if(LCDText[32-curHTTP.callbackPos] == ' ' || LCDText[32-curHTTP.callbackPos] ==
'\0')
        len -= TCPPutROMString(sktHTTP, (ROM BYTE*)"&nbsp;");
    else
        len -= TCPPut(sktHTTP, LCDText[32-curHTTP.callbackPos]);

    curHTTP.callbackPos--;
}
}

```

The initial call to `TCPisPutReady` determines how many bytes can be written to the buffer right now. The `TCPput` functions all return the number of bytes written, so we can subtract that value from `len` to track how much buffer space is left. When buffer space is exhausted, the function exits and waits to be called again. For subsequent calls, the value of `curHTTP.callbackPos` is exactly as we left it. The function resumes its output at that point.

Including Files

Often it is useful to include the entire contents of another file in your output. Most web pages have at least some portion that does not change, such as the header, menu of links, and footer. These sections can be abstracted out into separate files which makes them easier to manage and conserves storage space.

To include the entire contents of another file, use a dynamic variable that starts with `"inc:"`, such as `~inc:header.inc~`. This sequence will cause the file `header.inc` to be read from the file system and inserted at this location.

The following example indicates how to include a standard menu bar section into every page:

```
<div id="menu">~inc:menu.inc~/div>
```

At this time, dynamic variables are not recursive, so any variables located inside files included in this manner are not parsed.

1.9.1.9.1.2 HTTP2 Form Processing

Allows client input from HTML forms

Description

Many applications need to accept data from a user. A common solution is to present a form to the user in a web page, then have the device process the values submitted via this form. Web forms are usually submitted using one of two methods (**GET** and **POST**), and the HTTP2 web server supports both.

The GET Method

The GET method appends the data to the end of the URI. This data follows the question mark (?) in the browser's address bar. (ex: `http://mchpboard/form.htm?led1=0&led2=1&led3=0`) Data sent via GET is automatically decoded and stored in the `curHTTP.data` array. Since it is to be stored in memory, this data is limited to the size of `curHTTP.data`, which by default is 100 bytes. However, it is generally easier to process data received in this manner.

The callback function `HTTPExecuteGet` is implemented by the application developer to process this data and perform any necessary actions. The functions `HTTPGetArg` and `HTTPGetROMArg` provide an easy method to retrieve submitted values for processing.

The following example demonstrates a form to control several LEDs.

```

<form method="get" action="leds.htm">
  LED 1: <input type="checkbox" name="led1" value="1" /><br />
  LED 2: <input type="checkbox" name="led2" value="1" /><br />
  LED 3: <input type="checkbox" name="led3" value="1" /><br />
  <input type="submit" value="Set LEDs" />
</form>

```

Suppose a user selects the checkboxes for LED 1 and LED3. The following string will be submitted to the server:

```
GET /leds.htm?led1=1&led3=1 HTTP/1.1
```

The HTTP2 web server will parse this request and store the following string in `curHTTP.data`:

```
"led1\01\0led3\01\0\0"
```

It will then call `HTTPExecuteGet` to process this input. To process this data, that callback needs to do several things. First, it should call `MPFSGetFilename` to verify which form was submitted. (This step may be omitted if only one form is provided by the application.) Next, since a checkbox control was used a default state of unchecked must be assumed. Finally, the callback should search for each argument it expects, compare the value, and set the LED pins accordingly. The following example satisfies all these requirements:

```
HTTP_IO_RESULT HTTPExecuteGet(void)
{
    BYTE *ptr, filename[20];

    // Load the file name (filename[] must be large enough to hold
    // the longest file name expected)
    MPFSGetFilename(curHTTP.file, filename, 20);

    // Verify the file name
    if(!strcmppgm2ram(filename, (ROM char*)"leds.htm"))
    {
        // Assume a default state of off
        LED1_IO = 0;
        LED2_IO = 0;
        LED3_IO = 0;

        // Search for each LED parameter and process
        ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE*)"led1");
        if(ptr)
            LED1_IO = (*ptr == '1');

        ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE*)"led2");
        if(ptr)
            LED2_IO = (*ptr == '1');

        ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE*)"led3");
        if(ptr)
            LED3_IO = (*ptr == '1');
    }

    // Indicate completion
    return HTTP_IO_DONE;
}
```

The POST Method

The POST method transmits data after all the request headers have been sent. This data is not visible in the browser's address bar, and can only be seen with a packet capture tool. It does however use the same URL encoding method.

The HTTP2 server does not perform any pre-parsing of this data. All POST data is left in the TCP buffer, so the custom application will need to access the TCP buffer directly to retrieve and decode it. The functions `HTTPReadPostName` and `HTTPReadPostValue` have been provided to assist with these requirements. However, these functions can only be used when at least entire variables are expected to fit in the TCP buffer at once.

Most POST processing functions will be implemented as state machines in order to use these functions. The variable `curHTTP.smPost` is available to store the current state. This state machine variable is reset to zero with each new request. Functions should generally implement a state to read a variable name, and another to read an expected value. Additional states may be helpful depending on the application.

The following example form accepts an e-mail address, a subject, and a message body. Since this data will likely total over 100 bytes, it should be submitted via POST.

```
<form method="post" action="/email.htm">
  To: <input type="text" name="to" maxlength="50" /><br />
  Subject: <input type="text" name="subject" maxlength="50" /><br />
```

```

Message:<br />
<textarea name="msg" rows="6"></textarea><br />
<input type="submit" value="Send Message" /></div>
</form>

```

Suppose a user enters the following data into this form:

```

To: joe@picsaregood.com
Subject: Sent by a PIC
Message: I sent this message using my development board!

```

The HTTPExecutePost function will be called with the following data still in the TCP buffer:

```

to=joe%40picsaregood.com&subject=Sent+by+a+PIC
&msg=I+sent+this+message+using+my+development+board%21

```

To use the e-mail module, the application needs to read in the address and the subject, store those in RAM, then send the message. However, since the message is not guaranteed to fit in RAM all at once, it must be read as space is available and passed to the e-mail module. A state machine, coupled with the HTTPReadPostName and HTTPReadPostValue functions can simplify this greatly.

The following example callback function will properly parse this input. For this example, it is assumed that this is the only form the board accepts, so file name checking is not performed. The address will be stored at curHTTP.data[0:49], and the subject will be stored at curHTTP.data[50:99]. This is not the most optimal solution, but serves as a simple example.

```

HTTP_IO_RESULT HTTPExecutePost(void)
{
    BYTE *dest, temp[16];

    // Define state machine values
    #define SM_READ_NAME      (0u)
    #define SM_READ_VALUE    (1u)
    #define SM_START_MESSAGE (2u)
    #define SM_SEND_MESSAGE  (3u)

    switch(curHTTP.smPost)
    {
        case SM_READ_NAME:
            // Read the next variable name. If a complete name is
            // not found, request more data. This function will
            // automatically truncate invalid data to prevent
            // buffer overflows.
            if(HTTPReadPostName(temp,16) == HTTP_READ_INCOMPLETE)
                return HTTP_IO_NEED_DATA;

            // Save "to" values to curHTTP.data[0:49]
            if(!strcmppgm2ram((char*)temp, (ROM char*)"to"))
                dest = curHTTP.data;

            // Save "subject" values to curHTTP.data[50:99]
            else if(!strcmppgm2ram((char*)temp, (ROM char*)"subject"))
                dest = curHTTP.data + 50;

            // When a "msg" is encountered, start sending
            else if(!strcmppgm2ram((char*)temp, (ROM char*)"msg"))
            {
                curHTTP.smPost = SM_START_MESSAGE;
                break;
            }

            // Ignore unexpected values
            else
                dest = NULL;

            // Move to the next state, but do not break yet
            curHTTP.smPost = SM_READ_VALUE;

        case SM_READ_VALUE:
            // Read the next value. If a complete value is
            // not found, request more data. This function will
            // automatically truncate invalid data to prevent
            // buffer overflows.

```

```

    if(HTTPReadPostValue(dest,50) == HTTP_READ_INCOMPLETE)
        return HTTP_IO_NEED_DATA;

    // Return to read a new name
    curHTTP.smPost = SM_READ_NAME;
    break;

case SM_START_MESSAGE:
    // TODO: Perform necessary tasks to start sending the message.

    // Move on to sending the message body
    curHTTP.smPost = SM_SEND_MESSAGE;
    break;

case SM_SEND_MESSAGE:
    // The message may be longer than the TCP buffer can hold
    // at once. To avoid errors, read the data piece by
    // piece and send it to the e-mail module. This requires
    // using TCP functions directly.

    // Send all remaining data
    while(curHTTP.byteCount > 0)
    {
        // First check if data is ready
        if(TCPIsGetReady(sktHTTP) == 0)
            return HTTP_IO_NEED_DATA;

        // TODO: Read data with TCPGetArray and send
        // it to the e-mail module.
    }

    // Process is complete
    return HTTP_IO_DONE;
}

// Assume return for state machine convenience.
// Do not return HTTP_IO_NEED_DATA here by default, because
// doing so when more data will not arrive is cause for
// the HTTP2 server to return an error to the user.
return HTTP_IO_WAITING;
}

```

The previous example uses the HTTPReadPostName and HTTPReadPostValue functions, and also demonstrates using the need to use TCPIsGetReady, TCPGet, and TCPGetArray when longer values are expected. For applications that will receive and react to parameters immediately and have no need for a state machine, a simple while loop can be written around HTTPReadPostPair to accomplish the callback. The HTTPPostLCD function in the TCPIP Demo App provides a simple example of this.

For more examples, refer to custom_http_app.c in the TCPIP Demo App project.

1.9.1.9.1.3 HTTP2 Authentication

Verifies user names and passwords for access

Description

The HTTP protocol provides a method for servers to request a user name and password from a client before granting access to a page. The HTTP2 server supports this authentication mechanism, allowing developers to require valid credentials for access to some or all pages.

Authentication functionality is supported by two user-provided callback functions. The first, HTTPNeedsAuth, determines if the requested page requires valid credentials to proceed. The second, HTTPCheckAuth, checks the user name and password against an accepted list and determines whether to grant or deny access. This split between two callback functions is necessitated by the nature of the HTTP protocol and the low-memory architecture of the HTTP2 server. In cases where different credentials or sets of credentials may be accepted for different pages, the two functions communicate with each other through a single byte stored in curHTTP.isAuthorized.

Requiring Authentication

When a request is first made, the function HTTPNeedsAuth is called to determine if that page needs password protection. This function returns a value to instruct the HTTP2 server how to proceed. The most significant bit indicates whether or not access is granted. That is, values 0x80 and higher allow access unconditionally, while values 0x79 and lower will require a user name and password at a later point. The value returned is stored as `curHTTP.isAuthorized` so that it can be accessed by future callback functions.

The following example is the simplest case, in which all files require a password for access:

```
BYTE HTTPNeedsAuth(BYTE* cFile)
{
    return 0x00;
}
```

In some cases, only certain files will need to be protected. The second example requires a password for any file located in the `/treasure` folder:

```
BYTE HTTPNeedsAuth(BYTE* cFile)
{
    // Compare to "/treasure" folder. Don't use strcmp here, because
    // cFile has additional path info such as "/treasure/gold.htm"
    if(!memcmppgm2ram((void*)cFile, (ROM void*)"treasure", 8))
        return 0x00;

    return 0x80;
}
```

More complex uses could require an administrative user to access the `/admin` folder, while any authenticated user can access the rest of the site. The third example requires a different set of user name and password combinations for the `/admin` folder versus the rest of the site:

```
#define HTTP_AUTH_ADMIN      (0x00)
#define HTTP_AUTH_OTHER      (0x01)

BYTE HTTPNeedsAuth(BYTE* cFile)
{
    // Return a specific code for admin users
    if(!memcmppgm2ram((void*)cFile, (ROM void*)"admin", 5))
        return HTTP_AUTH_ADMIN;

    return HTTP_AUTH_OTHER;
}
```

Validating Credentials

The HTTPCheckAuth function determines if the supplied user name and password are valid to access this resource. Again, the most significant bit indicates whether or not access is granted. The value returned is also stored as `curHTTP.isAuthorized` so that it can be accessed by future callback functions.

The following example is the simplest case, in which one user/password pair is accepted for all pages:

```
BYTE HTTPCheckAuth(BYTE* cUser, BYTE* cPass)
{
    if(!strcmppgm2ram((char*)cUser, (ROM char*)"AliBaba") &&
        !strcmppgm2ram((char*)cPass, (ROM char*)"Open Sesame!"))
        return 0x80;

    return 0x00;
}
```

In some cases, you may have multiple users with various levels of access. The following example satisfies the needs used in the third example of HTTPNeedsAuth above:

```
BYTE HTTPCheckAuth(BYTE* cUser, BYTE* cPass)
{
    // Check for admin users first
    if(curHTTP.isAuthorized == HTTP_AUTH_ADMIN &&
        !strcmppgm2ram((char*)cUser, (ROM char*)"admin") &&
```

```

!strcmppgm2ram((char*)cPass, (ROM char*)"s3cREt") )
return 0x80;

if(!strcmppgm2ram((char*)cUser, (ROM char*)"kate") &&
!strcmppgm2ram((char*)cPass, (ROM char*)"puppies!") )
return 0x80;

return 0x00;
}

```

More complex uses are certainly feasible. Many applications may choose to store the user names and passwords in EEPROM or other non-volatile storage so that they may be updated by the end-user. Some applications may wish to return various values above 0x80 in HTTPCheckAuth so that later callback functions can determine which user logged in. The flexibility of these functions provides for many more possibilities that are not documented here but can be developed in just a few hours.

1.9.1.9.1.4 HTTP2 Cookies

Allows cookies to be set/retrieved, adding state to the HTTP protocol

Description

By design, HTTP is a session-less and state-less protocol; every connection is an independent session with no relation to another. Cookies were added to the protocol description to solve this problem. This feature allows a web server to store small bits of text in a user's browser. These values will be returned to the server with every request, allowing the server to associate session variables with a request. Cookies are typically used for more advanced authentication systems.

Best practice is generally to store the bulk of the data on the server, and store only a unique identifier with the browser. This cuts down on data overhead and ensures that the user cannot modify the values stored with the session. However, logic must be implemented in the server to expire old sessions and allocate memory for new ones. If sensitive data is being stored, it is also important that the identifier be random enough so as to prevent stealing or spoofing another user's cookies.

Retrieving Cookies

In the HTTP2 server, cookies are retrieved automatically. They are stored in `curHTTP.data`, just as any other GET form argument or URL parameter would be. The proper place to parse these values is therefore in the HTTPExecuteGet callback using the HTTPGetArg or HTTPGetROMArg functions to locate the values.

This model consumes some of the limited space available for URL parameters. Ensure that cookies do not consume more space than is available (as defined by `HTTP_MAX_DATA_LEN`) and that they will fit after any data that may be submitted via a GET form. If enough space is not available, the cookies will be truncated.

Setting Cookies

Cookies can be set in HTTPExecuteGet or HTTPExecutePost. To set a cookie, store the name/value pairs in `curHTTP.data` as a series of null-terminated strings. Then set, `curHTTP.hasArgs` equal to the number of name/value pairs to be set. For example, the following code sets a cookie indicating a user's preference for a type of cookie:

```

void HTTPExecuteGet(void)
{
    ...

    // Set a cookie
    strcpypgm2ram((char*)curHTTP.data, (ROM char*)"flavor\0oatmeal raisin");
    curHTTP.hasArgs = 1;

    ...
}

```

After this, all future requests from this browser will include the parameter "flavor" in `curHTTP.data`, along with the associated value of "oatmeal raisin".

1.9.1.9.1.5 HTTP2 Compression

Compresses static files for faster throughput

Description

All modern web browsers can receive files encoded with GZIP compression. For static files (those without dynamic variables), this can decrease the amount of data transmitted by as much as 60%.

The MPFS2 Utility will automatically determine which files can benefit from GZIP compression, and will store the compressed file in the MPFS2 image when possible. This generally includes all JavaScript and CSS files. (Images are typically already compressed, so the MPFS2 Utility will generally decide it is better to store them uncompressed.) This HTTP server will then seamlessly return this compressed file to the browser. Less non-volatile storage space will be required for the MPFS2 image, and faster transfers back to the client will result. No special configuration is required for this feature.

To prevent certain extensions from being compressed, use the Advanced Settings dialog in the MPFS2 Utility.



1.9.1.9.2 HTTP2 Public Members

Functions and variables accessible or implemented by the stack application

Enumerations

Name	Description
HTTP_IO_RESULT	Result states for execution callbacks
HTTP_READ_STATUS	Result states for HTTPPostReadName and HTTPPostReadValue

Functions

	Name	Description
	HTTPExecuteGet	Processes GET form field variables and cookies.
	HTTPExecutePost	Processes POST form variables and data.

Macros

Name	Description
HTTPReadPostPair	Reads a name and value pair from a URL encoded string in the TCP buffer.
sktHTTP	Access the current socket

Structures

Name	Description
HTTP_CONN	Stores extended state data for each connection

Variables

Name	Description
curHTTP	Current HTTP connection state

Description

The following functions and variables are accessible or implemented by the stack application.

1.9.1.9.2.1 curHTTP Variable**File**

http2.h

Syntax

```
HTTP_CONN curHTTP;
```

Description

Current HTTP connection state

1.9.1.9.2.2 HTTP_CONN Structure

File

http2.h

Syntax

```
typedef struct {
    uint32_t byteCount;
    uint32_t nextCallback;
    uint32_t callbackID;
    uint32_t callbackPos;
    uint8_t * ptrData;
    uint8_t * ptrRead;
    MPFS_HANDLE file;
    MPFS_HANDLE offsets;
    uint8_t hasArgs;
    uint8_t isAuthorized;
    HTTP_STATUS httpStatus;
    HTTP_FILE_TYPE fileType;
    uint8_t data[HTTP_MAX_DATA_LEN];
    uint8_t smPost;
} HTTP_CONN;
```

Members

Members	Description
uint32_t byteCount;	How many bytes have been read so far
uint32_t nextCallback;	Byte index of the next callback
uint32_t callbackID;	Callback ID to execute, also used as watchdog timer
uint32_t callbackPos;	Callback position indicator
uint8_t * ptrData;	Points to first free byte in data
uint8_t * ptrRead;	Points to current read location
MPFS_HANDLE file;	File pointer for the file being served
MPFS_HANDLE offsets;	File pointer for any offset info being used
uint8_t hasArgs;	True if there were get or cookie arguments
uint8_t isAuthorized;	0x00-0x79 on fail, 0x80-0xff on pass
HTTP_STATUS httpStatus;	Request method/status
HTTP_FILE_TYPE fileType;	File type to return with Content-Type
uint8_t data[HTTP_MAX_DATA_LEN];	General purpose data buffer
uint8_t smPost;	POST state machine variable

Description

Stores extended state data for each connection

1.9.1.9.2.3 HTTP_IO_RESULT Enumeration

File

http2.h

Syntax

```
typedef enum {
    HTTP_IO_DONE = 0u,
    HTTP_IO_NEED_DATA,
    HTTP_IO_WAITING
} HTTP_IO_RESULT;
```

Members

Members	Description
HTTP_IO_DONE = 0u	Finished with procedure

HTTP_IO_NEED_DATA	More data needed to continue, call again later
HTTP_IO_WAITING	Waiting for asynchronous process to complete, call again later

Description

Result states for execution callbacks

1.9.1.9.2.4 HTTP_READ_STATUS Enumeration**File**

http2.h

Syntax

```
typedef enum {
    HTTP_READ_OK = 0u,
    HTTP_READ_TRUNCATED,
    HTTP_READ_INCOMPLETE
} HTTP_READ_STATUS;
```

Members

Members	Description
HTTP_READ_OK = 0u	Read was successful
HTTP_READ_TRUNCATED	Buffer overflow prevented by truncating value
HTTP_READ_INCOMPLETE	Entire object is not yet in the buffer. Try again later.

Description

Result states for HTTPPostReadName and HTTPPostReadValue

1.9.1.9.2.5 HTTPExecuteGet Function

Processes GET form field variables and cookies.

File

http2.h

Syntax

```
HTTP_IO_RESULT HTTPExecuteGet();
```

Description

This function is implemented by the application developer in custom_http_app.c. Its purpose is to parse the data received from URL parameters (GET method forms) and cookies and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated.

When this function is called, curHTTP.data contains sequential name/value pairs of strings representing the data received. In this format, HTTPGetArg and HTTPGetROMArg can be used to search for specific variables in the input. If data buffer space associated with this connection is required, curHTTP.data may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including HTTPExecutePost and any HTTPPrint_varname dynamic substitutions.

This function may also issue redirections by setting curHTTP.data to the destination file name or URL, and curHTTP.httpStatus to HTTP_REDIRECT.

Finally, this function may set cookies. Set curHTTP.data to a series of name/value string pairs (in the same format in which parameters arrive) and then set curHTTP.hasArgs equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in curHTTP.data.

Remarks

This function is only called if variables are received via URL parameters or Cookie arguments. This function may NOT write to the TCP buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use `curHTTP.callbackPos` or `curHTTP.data` for storage associated with individual requests.

Preconditions

None

Return Values

Return Values	Description
HTTP_IO_DONE	application is done processing
HTTP_IO_NEED_DATA	this value may not be returned because more data will not become available
HTTP_IO_WAITING	the application is waiting for an asynchronous process to complete, and this function should be called again later

Function

HTTP_IO_RESULT HTTPExecuteGet(void)

1.9.1.9.2.6 HTTPExecutePost Function

Processes POST form variables and data.

File

http2.h

Syntax

HTTP_IO_RESULT HTTPExecutePost () ;

Description

This function is implemented by the application developer in `custom_http_app.c`. Its purpose is to parse the data received from POST forms and perform any application-specific tasks in response to these inputs. Any required authentication has already been validated before this function is called.

When this function is called, POST data will be waiting in the TCP buffer. `curHTTP.byteCount` will indicate the number of bytes remaining to be received before the browser request is complete.

Since data is still in the TCP buffer, the application must call `TCPGet` or `TCPGetArray` in order to retrieve bytes. When this is done, `curHTTP.byteCount` MUST be updated to reflect how many bytes now remain. The functions `TCPFind`, `TCPFindString`, `TCPFindROMString`, `TCPFindArray`, and `TCPFindROMArray` may be helpful to locate data in the TCP buffer.

In general, data submitted from web forms via POST is URL encoded. The `HTTPURLDecode` function can be used to decode this information back to a standard string if required. If data buffer space associated with this connection is required, `curHTTP.data` may be overwritten here once the application is done with the values. Any data placed there will be available to future callbacks for this connection, including `HTTPExecutePost` and any `HTTPPrint_varname` dynamic substitutions.

Whenever a POST form is processed it is recommended to issue a redirect back to the browser, either to a status page or to the same form page that was posted. This prevents accidental duplicate submissions (by clicking refresh or back/forward) and avoids browser warnings about "resubmitting form data". Redirects may be issued to the browser by setting `curHTTP.data` to the destination file or URL, and `curHTTP.httpStatus` to `HTTP_REDIRECT`.

Finally, this function may set cookies. Set `curHTTP.data` to a series of name/value string pairs (in the same format in which parameters arrive) and then set `curHTTP.hasArgs` equal to the number of cookie name/value pairs. The cookies will be transmitted to the browser, and any future requests will have those values available in `curHTTP.data`.

Remarks

This function is only called when the request method is POST, and is only used when `HTTP_USE_POST` is defined. This method may NOT write to the TCP buffer.

This function may service multiple HTTP requests simultaneously. Exercise caution when using global or static variables inside this routine. Use `curHTTP.callbackPos` or `curHTTP.data` for storage associated with individual requests.

Preconditions

None

Return Values

Return Values	Description
HTTP_IO_DONE	application is done processing
HTTP_IO_NEED_DATA	more data is needed to continue, and this function should be called again later
HTTP_IO_WAITING	the application is waiting for an asynchronous process to complete, and this function should be called again later

Function

HTTP_IO_RESULT HTTPExecutePost(void)

1.9.1.9.2.7 HTTPReadPostPair Macro

Reads a name and value pair from a URL encoded string in the TCP buffer.

File

http2.h

Syntax

```
#define HTTPReadPostPair(cData, wLen) HTTPReadPostValue(cData, wLen)
```

Description

Reads a name and value pair from a URL encoded string in the TCP buffer. This function is meant to be called from an HTTPExecutePost callback to facilitate easier parsing of incoming data. This function also prevents buffer overflows by forcing the programmer to indicate how many bytes are expected. At least 2 extra bytes are needed in cData over the maximum length of data expected to be read.

This function will read until the next '&' character, which indicates the end of a value parameter. It assumes that the front of the buffer is the beginning of the name parameter to be read.

This function properly updates curHTTP.byteCount by decrementing it by the number of bytes read. It also removes the delimiting '&' from the buffer.

Once complete, two strings will exist in the cData buffer. The first is the parameter name that was read, while the second is the associated value.

Remarks

This function is aliased to HTTPReadPostValue, since they effectively perform the same task. The name is provided only for completeness.

Preconditions

Front of TCP buffer is the beginning of a name parameter, and the rest of the TCP buffer contains a URL-encoded string with a name parameter terminated by a '=' character and a value parameter terminated by a '&'.

Parameters

Parameters	Description
cData	where to store the name and value strings once they are read
wLen	how many bytes can be written to cData

Return Values

Return Values	Description
HTTP_READ_OK	name and value were successfully read
HTTP_READ_TRUNCATED	entire name and value could not fit in the buffer, so input was truncated and data has been lost

HTTP_READ_INCOMPLETE	entire name and value was not yet in the buffer, so call this function again later to retrieve
----------------------	--

Function

```
HTTP_READ_STATUS HTTPReadPostPair(uint8_t *cData, uint16_t wLen)
```

1.9.1.9.2.8 sktHTTP Macro

File

```
http2.h
```

Syntax

```
#define sktHTTP httpStubs[curHTTPID].socket // Access the current socket
```

Description

Access the current socket

1.9.1.9.3 HTTP2 Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
☞	HTTPInit	
☞	HTTPServer	This is function HTTPServer.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.9.3.1 HTTPInit Function

File

```
http2.h
```

Syntax

```
void HTTPInit();
```

Section

Function Prototypes

1.9.1.9.3.2 HTTPServer Function

File

```
http2.h
```

Syntax

```
void HTTPServer();
```

Description

This is function HTTPServer.


1.9.1.9.4 HTTP2 Internal Members

Functions and variables used internally by the HTTP2 module

Enumerations

Name	Description
HTTP_FILE_TYPE	File type definitions
HTTP_STATUS	Supported Commands and Server Response Codes
SM_HTTP2	Basic HTTP Connection State Machine

Functions

	Name	Description
	HTTPGetROMArg	Non-ROM variant for C30 / C32

Macros

Name	Description
HTTP_CACHE_LEN	Max lifetime (sec) of static responses as string
HTTP_MAX_DATA_LEN	This is macro HTTP_MAX_DATA_LEN.
HTTP_MIN_CALLBACK_FREE	This is macro HTTP_MIN_CALLBACK_FREE.
HTTP_PORT	Listening port for HTTP server
HTTP_TIMEOUT	Max time (sec) to await more data before timing out and disconnecting the socket
HTTPS_PORT	Listening port for HTTPS server (when SSL enabled)
RESERVED_HTTP_MEMORY	Macro indicating how much RAM to allocate on an Ethernet controller to store HTTP state data.

Structures

Name	Description
HTTP_STUB	HTTP Connection Struct Stores partial state data for each connection Meant for storage in fast access RAM

Variables

Name	Description
curHTTPID	This is variable curHTTPID.
httpStubs	This is variable httpStubs.

Description

The following functions and variables are designated as internal to the HTTP2 module.

1.9.1.9.4.1 curHTTPID Variable**File**

http2.h

Syntax

```
uint8_t curHTTPID;
```

Description

This is variable curHTTPID.

1.9.1.9.4.2 HTTP_CACHE_LEN Macro**File**

http2.h

Syntax

```
#define HTTP_CACHE_LEN ("600") // Max lifetime (sec) of static responses as string
```

Description

Max lifetime (sec) of static responses as string

1.9.1.9.4.3 HTTP_FILE_TYPE Enumeration

File

http2.h

Syntax

```
typedef enum {
    HTTP_TXT = 0u,
    HTTP_HTM,
    HTTP_HTML,
    HTTP_CGI,
    HTTP_XML,
    HTTP_CSS,
    HTTP_GIF,
    HTTP_PNG,
    HTTP_JPG,
    HTTP_JAVA,
    HTTP_WAV,
    HTTP_UNKNOWN
} HTTP_FILE_TYPE;
```

Members

Members	Description
HTTP_TXT = 0u	File is a text document
HTTP_HTM	File is HTML (extension .htm)
HTTP_HTML	File is HTML (extension .html)
HTTP_CGI	File is HTML (extension .cgi)
HTTP_XML	File is XML (extension .xml)
HTTP_CSS	File is stylesheet (extension .css)
HTTP_GIF	File is GIF image (extension .gif)
HTTP_PNG	File is PNG image (extension .png)
HTTP_JPG	File is JPG image (extension .jpg)
HTTP_JAVA	File is java (extension .java)
HTTP_WAV	File is audio (extension .wav)
HTTP_UNKNOWN	File type is unknown

Description

File type definitions

1.9.1.9.4.4 HTTP_MAX_DATA_LEN Macro

File

http2.h

Syntax

```
#define HTTP_MAX_DATA_LEN (100u)
```

Description

This is macro HTTP_MAX_DATA_LEN.

1.9.1.9.4.5 HTTP_MIN_CALLBACK_FREE Macro

File

http2.h

Syntax

```
#define HTTP_MIN_CALLBACK_FREE (16u)
```

Description

This is macro HTTP_MIN_CALLBACK_FREE.

1.9.1.9.4.6 HTTP_PORT Macro**File**

http2.h

Syntax

```
#define HTTP_PORT (80u)    // Listening port for HTTP server
```

Description

Listening port for HTTP server

1.9.1.9.4.7 HTTP_STATUS Enumeration**File**

http2.h

Syntax

```
typedef enum {
    HTTP_GET = 0u,
    HTTP_POST,
    HTTP_BAD_REQUEST,
    HTTP_UNAUTHORIZED,
    HTTP_NOT_FOUND,
    HTTP_OVERFLOW,
    HTTP_INTERNAL_SERVER_ERROR,
    HTTP_NOT_IMPLEMENTED,
    HTTP_MPFS_FORM,
    HTTP_MPFS_UP,
    HTTP_MPFS_OK,
    HTTP_MPFS_ERROR,
    HTTP_REDIRECT,
    HTTP_SSL_REQUIRED
} HTTP_STATUS;
```

Members

Members	Description
HTTP_GET = 0u	GET command is being processed
HTTP_POST	POST command is being processed
HTTP_BAD_REQUEST	400 Bad Request will be returned
HTTP_UNAUTHORIZED	401 Unauthorized will be returned
HTTP_NOT_FOUND	404 Not Found will be returned
HTTP_OVERFLOW	414 Request-URI Too Long will be returned
HTTP_INTERNAL_SERVER_ERROR	500 Internal Server Error will be returned
HTTP_NOT_IMPLEMENTED	501 Not Implemented (not a GET or POST command)
HTTP_MPFS_FORM	Show the MPFS Upload form
HTTP_MPFS_UP	An MPFS Upload is being processed
HTTP_MPFS_OK	An MPFS Upload was successful
HTTP_MPFS_ERROR	An MPFS Upload was not a valid image
HTTP_REDIRECT	302 Redirect will be returned
HTTP_SSL_REQUIRED	403 Forbidden is returned, indicating SSL is required

Description

Supported Commands and Server Response Codes

1.9.1.9.4.8 HTTP_STUB Structure

File

http2.h

Syntax

```
typedef struct {
    SM_HTTP2 sm;
    TCP_SOCKET socket;
} HTTP_STUB;
```

Members

Members	Description
SM_HTTP2 sm;	Current connection state
TCP_SOCKET socket;	Socket being served

Description

HTTP Connection Struct Stores partial state data for each connection Meant for storage in fast access RAM

1.9.1.9.4.9 HTTP_TIMEOUT Macro

File

http2.h

Syntax

```
#define HTTP_TIMEOUT (45u) // Max time (sec) to await more data before timing out and disconnecting the socket
```

Description

Max time (sec) to await more data before timing out and disconnecting the socket

1.9.1.9.4.10 HTTPGetROMArg Function

File

http2.h

Syntax

```
uint8_t * HTTPGetROMArg(uint8_t * cData, ROM uint8_t * cArg);
```

Description

Non-ROM variant for C30 / C32

1.9.1.9.4.11 HTTPS_PORT Macro

File

http2.h

Syntax

```
#define HTTPS_PORT (443u) // Listening port for HTTPS server (when SSL enabled)
```

Description

Listening port for HTTPS server (when SSL enabled)

1.9.1.9.4.12 httpStubs Variable

File

http2.h

Syntax

```
HTTP_STUB httpStubs[MAX_HTTP_CONNECTIONS];
```

Description

This is variable httpStubs.

1.9.1.9.4.13 SM_HTTP2 Enumeration

File

http2.h

Syntax

```
typedef enum {
    SM_HTTP_IDLE = 0u,
    SM_HTTP_PARSE_REQUEST,
    SM_HTTP_PARSE_HEADERS,
    SM_HTTP_AUTHENTICATE,
    SM_HTTP_PROCESS_GET,
    SM_HTTP_PROCESS_POST,
    SM_HTTP_PROCESS_REQUEST,
    SM_HTTP_SERVE_HEADERS,
    SM_HTTP_SERVE_COOKIES,
    SM_HTTP_SERVE_BODY,
    SM_HTTP_SEND_FROM_CALLBACK,
    SM_HTTP_DISCONNECT
} SM_HTTP2;
```

Members

Members	Description
SM_HTTP_IDLE = 0u	Socket is idle
SM_HTTP_PARSE_REQUEST	Parses the first line for a file name and GET args
SM_HTTP_PARSE_HEADERS	Reads and parses headers one at a time
SM_HTTP_AUTHENTICATE	Validates the current authorization state
SM_HTTP_PROCESS_GET	Invokes user callback for GET args or cookies
SM_HTTP_PROCESS_POST	Invokes user callback for POSTed data
SM_HTTP_PROCESS_REQUEST	Begins the process of returning data
SM_HTTP_SERVE_HEADERS	Sends any required headers for the response
SM_HTTP_SERVE_COOKIES	Adds any cookies to the response
SM_HTTP_SERVE_BODY	Serves the actual content
SM_HTTP_SEND_FROM_CALLBACK	Invokes a dynamic variable callback
SM_HTTP_DISCONNECT	Disconnects the server and closes all files

Description

Basic HTTP Connection State Machine

1.9.1.9.4.14 RESERVED_HTTP_MEMORY Macro

File

mac.h

Syntax

```
#define RESERVED_HTTP_MEMORY 0u1
```

Description

Macro indicating how much RAM to allocate on an Ethernet controller to store HTTP state data.

1.9.1.10 ICMP

Provides Ping functionality.

Description

The Internet Control Message Protocol is used to send error and status messages and requests. The ICMP module implements the Echo Reply message type (commonly referred to as a ping) which can be used to determine if a specified host is reachable across an IP network from a device running the TCP/IP stack. An ICMP server is also supported to respond to pings from other devices.

1.9.1.10.1 ICMP Public Members

Functions and variables accessible or implemented by the stack application

Functions

	Name	Description
≡	ICMPBeginUsage	This is function ICMPBeginUsage.
≡	ICMPGetReply	This is function ICMPGetReply.
≡	ICMPEndUsage	This is function ICMPEndUsage.

Macros

Name	Description
ICMPSendPingToHostROM	This is macro ICMPSendPingToHostROM.

Description

The following functions and variables are accessible or implemented by the stack application.

1.9.1.10.1.1 ICMPBeginUsage Function

File

icmp.h

Syntax

```
bool ICMPBeginUsage();
```

Description

This is function ICMPBeginUsage.

1.9.1.10.1.2 ICMPGetReply Function

File

icmp.h

Syntax

```
int32_t ICMPGetReply();
```

Description

This is function ICMPGetReply.

1.9.1.10.1.3 ICMPEndUsage Function

File

icmp.h

Syntax

```
void ICMPEndUsage();
```

Description

This is function ICMPEndUsage.

1.9.1.10.1.4 ICMPSendPingToHostROM Macro

File

icmp.h

Syntax

```
#define ICMPSendPingToHostROM(a) ICMPSendPingToHost((uint8_t *) (a))
```

Description

This is macro ICMPSendPingToHostROM.

1.9.1.10.2 ICMP Internal Members

Functions and variables used internally by the ICMP module

Description

The following functions and variables are designated as internal to the ICMP module.

1.9.1.11 MPFS2

Provides a light-weight file system.

Description

The MPFS2 file system module provides a light-weight read-only file system that can be stored in external EEPROM, external serial Flash, or internal Flash program memory. This file system serves as the basis for the HTTP2 web server module, but is also used by the SNMP module and is available to other applications that require basic read-only storage capabilities.

The MPFS2 module includes an MPFS2 Utility that runs on your PC. This program builds MPFS2 images in various formats for use in the supported storage mediums. More information is available in the MPFS2 Utility section.

Using External Storage

For external storage, the MPFS2 file system supports Microchip 25LCxxx EEPROM parts for densities up to 1Mbit. SST 25VFxxxB serial Flash parts are also supported for densities up to 32Mbit.

To use external EEPROM storage, ensure that the configuration macro MPFS_USE_EEPROM is defined in tcpip_config.h. If you are using a 1Mbit part (25LC1024), also be sure to define USE_EEPROM_25LC1024 to enable the 24-bit device addressing used by that part. For external serial Flash, define MPFS_USE_SPI_FLASH instead of the EEPROM macros.

Images stored externally are uploaded via HTTP. This can be accomplished using the MPFS2 Utility, or can be accessed directly from a browser. Uploading files directly is described in the MPFS2 Utility documentation. Uploading images via HTTP can be accomplished as described in the Getting Started section.

When storing images externally, space can be reserved for separate application use. The configuration macro

MPFS_RESERVE_BLOCK controls the size of this space. The specified number of bytes will be reserved at the beginning address of the storage device (0x000000). When using serial Flash, this address must be a multiple of the flash sector size (4096 bytes).

Using Internal Flash Storage

When storing images in internal Flash program memory, new images cannot be uploaded at run time. Instead, the image is compiled in as part of your project in the MPLABX IDE. To select this storage option comment out the configuration macro MPFS_USE_EEPROM in `tcpip_config.h`, then ensure that the image file generated by the MPFS2 Utility is included in the MPLABX project.

Other Considerations

MPFS2 defines a fixed number of files that can be opened simultaneously. The configuration parameter MAX_MPFS_HANDLES controls how many files can be opened at once. If this resource is depleted, no new files can be opened until MPFSClose is called for an existing handle. The HTTP2 web server expects to be able to use at least two handles for each connection, plus one extra. Additional handles should be allocated if other modules will be accessing the file system as well.

1.9.1.11.1 MPFS2 Public Members

Functions and variables accessible by the stack application

Enumerations

Name	Description
MPFS_SEEK_MODE	Indicates the method for MPFSSeek

Functions

	Name	Description
≡	MPFSClose	This is function MPFSClose.
≡	MPFSFormat	This is function MPFSFormat.
≡	MPFSGetBytesRem	This is function MPFSGetBytesRem.
≡	MPFSGetEndAddr	This is function MPFSGetEndAddr.
≡	MPFSGetFlags	This is function MPFSGetFlags.
≡	MPFSGetID	This is function MPFSGetID.
≡	MPFSGetMicrotime	This is function MPFSGetMicrotime.
≡	MPFSGetPosition	This is function MPFSGetPosition.
≡	MPFSGetSize	This is function MPFSGetSize.
≡	MPFSGetStartAddr	This is function MPFSGetStartAddr.
≡	MPFSGetTimestamp	This is function MPFSGetTimestamp.

Macros

Name	Description
MPFS_INVALID	Indicates a position pointer is invalid
MPFS_INVALID_HANDLE	Indicates that a handle is not valid

Types

Name	Description
MPFS_HANDLE	MPFS Handles are currently stored as BYTES

Description

The following functions and variables are accessible by the stack application.

1.9.1.11.1.1 MPFS_HANDLE Type

File

mpfs2.h

Syntax

```
typedef uint8_t MPFS_HANDLE;
```

Description

MPFS Handles are currently stored as BYTES

1.9.1.11.1.2 MPFS_INVALID Macro

File

mpfs2.h

Syntax

```
#define MPFS_INVALID (0xfffffffffu) // Indicates a position pointer is invalid
```

Description

Indicates a position pointer is invalid

1.9.1.11.1.3 MPFS_INVALID_HANDLE Macro

File

mpfs2.h

Syntax

```
#define MPFS_INVALID_HANDLE (0xffu) // Indicates that a handle is not valid
```

Description

Indicates that a handle is not valid

1.9.1.11.1.4 MPFS_SEEK_MODE Enumeration

File

mpfs2.h

Syntax

```
typedef enum {
    MPFS_SEEK_START = 0u,
    MPFS_SEEK_END,
    MPFS_SEEK_FORWARD,
    MPFS_SEEK_REWIND
} MPFS_SEEK_MODE;
```

Members

Members	Description
MPFS_SEEK_START = 0u	Seek forwards from the front of the file
MPFS_SEEK_END	Seek backwards from the end of the file
MPFS_SEEK_FORWARD	Seek forward from the current position
MPFS_SEEK_REWIND	See backwards from the current position

Description

Indicates the method for MPFSSeek

1.9.1.11.1.5 MPFSClose Function

File

mpfs2.h

Syntax

```
void MPFSClose(MPFS_HANDLE hMPFS);
```

Description

This is function MPFSClose.

1.9.1.11.1.6 MPFSFormat Function

File

mpfs2.h

Syntax

```
MPFS_HANDLE MPFSFormat();
```

Description

This is function MPFSFormat.

1.9.1.11.1.7 MPFSGetBytesRem Function

File

mpfs2.h

Syntax

```
uint32_t MPFSGetBytesRem(MPFS_HANDLE hMPFS);
```

Description

This is function MPFSGetBytesRem.

1.9.1.11.1.8 MPFSGetEndAddr Function

File

mpfs2.h

Syntax

```
MPFS_PTR MPFSGetEndAddr(MPFS_HANDLE hMPFS);
```

Description

This is function MPFSGetEndAddr.

1.9.1.11.1.9 MPFSGetFlags Function

File

mpfs2.h

Syntax

```
uint16_t MPFSGetFlags(MPFS_HANDLE hMPFS);
```

Description

This is function MPFSGetFlags.

1.9.1.11.1.10 MPFSGetID Function

File

mpfs2.h

Syntax

```
uint16_t MPFSGetID(MPFS_HANDLE hMPFS);
```

Description

This is function MPFSGetID.

1.9.1.11.1.11 MPFSGetMicrotime Function

File

mpfs2.h

Syntax

```
uint32_t MPFSGetMicrotime(MPFS_HANDLE hMPFS);
```

Description

This is function MPFSGetMicrotime.

1.9.1.11.1.12 MPFSGetPosition Function

File

mpfs2.h

Syntax

```
uint32_t MPFSGetPosition(MPFS_HANDLE hMPFS);
```

Description

This is function MPFSGetPosition.

1.9.1.11.1.13 MPFSGetSize Function

File

mpfs2.h

Syntax

```
uint32_t MPFSGetSize(MPFS_HANDLE hMPFS);
```

Description

This is function MPFSGetSize.

1.9.1.11.1.14 MPFSGetStartAddr Function

File

mpfs2.h

Syntax

```
MPFS_PTR MPFSGetStartAddr(MPFS_HANDLE hMPFS);
```

Description

This is function MPFSGetStartAddr.

1.9.1.11.1.15 MPFSGetTimestamp Function

File

mpfs2.h

Syntax

```
uint32_t MPFSGetTimestamp(MPFS_HANDLE hMPFS);
```

Description

This is function MPFSGetTimestamp.

1.9.1.11.2 MPFS2 Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
	MPFSInit	

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.11.2.1 MPFSInit Function

File

mpfs2.h

Syntax

```
void MPFSInit();
```


Section

Function Definitions

1.9.1.11.3 MPFS2 Internal Members

Functions and variables internal to the MPFS2 module

Functions

	Name	Description
	MPFSOpenROM	Non-ROM variant for C30 / C32

Macros

Name	Description
MPFS_WRITE_PAGE_SIZE	Defines the size of a page in EEPROM
MPFS2_FLAG_HASINDEX	Indicates a file has an associated index of dynamic variables
MPFS2_FLAG_ISZIPPED	Indicates a file is compressed with GZIP compression
MPFSTell	Alias of MPFSGetPosition
MPFS_INVALID_FAT	Indicates an invalid FAT cache

Structures

Name	Description
MPFS_STUB	Stores each file handle's information Handles are free when addr = MPFS_INVALID

MPFS_FAT_RECORD	Stores the data for an MPFS2 FAT record
-----------------	---

Types

Name	Description
MPFS_PTR	MPFS Pointers are currently DWORDs

Description

The following functions and variables are designated as internal to the MPFS2 module.

1.9.1.11.3.1 MPFS_PTR Type**File**

mpfs2.h

Syntax

```
typedef uint32_t MPFS_PTR;
```

Description

MPFS Pointers are currently DWORDs

1.9.1.11.3.2 MPFS_STUB Structure**File**

mpfs2.h

Syntax

```
typedef struct {
    MPFS_PTR addr;
    uint32_t bytesRem;
    uint16_t fatID;
} MPFS_STUB;
```

Members

Members	Description
MPFS_PTR addr;	Current address in the file system
uint32_t bytesRem;	How many bytes remain in this file
uint16_t fatID;	ID of which file in the FAT was accessed

Description

Stores each file handle's information Handles are free when addr = MPFS_INVALID

1.9.1.11.3.3 MPFS_WRITE_PAGE_SIZE Macro**File**

mpfs2.h

Syntax

```
#define MPFS_WRITE_PAGE_SIZE (64u) // Defines the size of a page in EEPROM
```

Description

Defines the size of a page in EEPROM

1.9.1.11.3.4 MPFS2_FLAG_HASINDEX Macro**File**

mpfs2.h

Syntax

```
#define MPFS2_FLAG_HASINDEX ((uint16_t)0x0002) // Indicates a file has an associated index of dynamic variables
```

Description

Indicates a file has an associated index of dynamic variables

1.9.1.11.3.5 MPFS2_FLAG_ISZIPPED Macro**File**

mpfs2.h

Syntax

```
#define MPFS2_FLAG_ISZIPPED ((uint16_t)0x0001) // Indicates a file is compressed with GZIP compression
```

Description

Indicates a file is compressed with GZIP compression

1.9.1.11.3.6 MPFSOpenROM Function**File**

mpfs2.h

Syntax

```
MPFS_HANDLE MPFSOpenROM(ROM uint8_t * cFile);
```

Description

Non-ROM variant for C30 / C32

1.9.1.11.3.7 MPFSTell Macro**File**

mpfs2.h

Syntax

```
#define MPFSTell(a) MPFSGetPosition(a)
```

Description

Alias of MPFSGetPosition

1.9.1.11.3.8 MPFS_FAT_RECORD Structure**File**

mpfs2.h

Syntax

```
typedef struct {  
    uint32_t string;  
    uint32_t data;  
    uint32_t len;  
    uint32_t timestamp;  
    uint32_t microtime;  
    uint16_t flags;  
} MPFS_FAT_RECORD;
```

Members

Members	Description
uint32_t string;	Pointer to the file name
uint32_t data;	Address of the file data
uint32_t len;	Length of file data
uint32_t timestamp;	Timestamp of file
uint32_t microtime;	Microtime stamp of file
uint16_t flags;	Flags for this file

Description

Stores the data for an MPFS2 FAT record

1.9.1.11.3.9 MPFS_INVALID_FAT Macro**File**

mpfs2.h

Syntax

```
#define MPFS_INVALID_FAT (0xffffu) // Indicates an invalid FAT cache
```

Description

Indicates an invalid FAT cache

1.9.1.12 NBNS

Describes the NetBIOS Name Service protocol.


Description

The NetBIOS Name Service protocol associates host names with IP addresses, similarly to DNS, but on the same IP subnet. Practically, this allows the assignment of human-name hostnames to access boards on the same subnet. For example, in the "TCPIP Demo App" demonstration project, the demo board is programmed with the human name 'mchpboard' so it can be accessed directly instead of with its IP address.

1.9.1.12.1 NBNS Stack Members

Functions and variables intended to be accessed only by the stack.

Functions

	Name	Description
	NBNSTask	This is function NBNSTask.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.12.1.1 NBNSTask Function**File**

nbns.h

Syntax

```
void NBNSTask( ) ;
```

Description

This is function NBNSTask.

1.9.1.13 Performance Tests

Tests TCP and UDP performance of an application.

Description



The TCP and UDP Performance Test modules provide a method for obtaining metrics about the stack's performance. They can be used to benchmark the stack on various hardware platforms, and are also useful to determine how your custom application has affected stack performance.

The stack automatically calls these modules when they are included, so you never need to call any functions directly. Instructions for use of the modules can be found in the documentation for UDPPerformanceTask, TCPTXPerformanceTask, and TCPRXPerformanceTask.

1.9.1.13.1 Performance Test Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
	TCPPerformanceTask	This is function TCPPerformanceTask.
	UDPPerformanceTask	This is function UDPPerformanceTask.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.13.1.1 TCPPerformanceTask Function

File

tcp_performance_test.h

Syntax

`void TCPPerformanceTask() ;`

Description

This is function TCPPerformanceTask.

1.9.1.13.1.2 UDPPerformanceTask Function

File

udp_performance_test.h

Syntax

`void UDPPerformanceTask() ;`

Description

This is function UDPPerformanceTask.

1.9.1.13.2 Performance Test Internal Members

Functions and variables internal to the module

Description

The following functions and variables are designated as internal to the module.

1.9.1.14 SMTP Client

Sends e-mail messages across the internet.

Description

The SMTP client module in the TCP/IP Stack lets applications send e-mails to any recipient worldwide. These messages could include status information or important alerts. Using the e-mail to SMS gateways provided by most cell phone carriers, these messages can also be delivered directly to cell phone handsets.

Using the SMTP client requires access to a local mail server (such as mail.yourdomain.com) for reliable operation. Your ISP or network administrator can provide the correct address, but end-user applications will need an interface to provide this data.

1.9.1.14.1 SMTP Client Examples

Examples for using the SMTP Client

Description

The following two examples demonstrate the use of the SMTP client in different scenarios. The first, and simpler example, sends a short message whose contents are all located in RAM at once.

The second example is more involved and demonstrates generating a message on the fly in the case where the entire message cannot fit into RAM at once. In this case, the message is started by the stack, but the delivery of the contents happens in pieces and is handled by the application.

1.9.1.14.1.1 SMTP Client Short Message Example

Send a message whose contents can be read from memory

Description

The SMTP client API is simplified when messages can be buffered entirely in RAM. The `SMTPDemo` example provided in `main.c` sends a brief e-mail message indicating the current status of the board's buttons. This document will walk through that example.

Make sure `STACK_USE_SMTP_CLIENT` is uncommented in `TCPConfig.h` before continuing.

The diagram below provides an overview of the process:

First, call `SMTPBeginUsage` to verify that the SMTP client is available and to begin a new message. If `FALSE` is returned, the SMTP client is busy and the application must return to the main loop to allow `StackTask` to execute again.

Next, set the local relay server to use as `SMTPClient.Server`. If the local relay server requires a user name and password, set `SMTPClient.Username` and `SMTPClient.Password` to the appropriate credentials.

If server parameters are not set, the stack will attempt to deliver the message directly to its destination host. This will likely fail due to spam prevention measures put in place by most ISPs and network administrators.

Continue on to set the header strings as necessary for the message. This includes the subject line, from address, and any recipients you need to add. Finally, set `SMTPClient.Body` to the message to be sent.

At this point, verify that `SMTPClient.ROMPointers` is correctly configured for any strings that are stored in program memory. Once the message is ready to send, call `SMTPSendMail` to instruct the SMTP client to begin transmission.

The application must now call `SMTPIsBusy` until it returns `FALSE`. Each time `TRUE` is returned, return to the main loop and wait for `StackTask` to execute again. This allows the SMTP server to continue its work in a cooperative multitasking manner. Once `FALSE` is returned, call `SMTPEndUsage` to release the SMTP client. Check the return value of this function to determine if the message was successfully sent.

The example in `main.c` needs minor modifications to use your e-mail address. The `Server` and `To` fields must be set in `SMTPDemo` in order for the message to be properly delivered. Once this is done, holding down `BUTTON2` and `BUTTON3` simultaneously (the left-most two buttons) will begin sending the message. `LED1` will light as the message is being processed, and will extinguish when the SMTP state machine completes. If the transmission was successful `LED2` will light, otherwise it will remain dark.

1.9.1.14.1.2 SMTP Client Long Message Example

Sends a long message built in pieces by the application

Description

The SMTP client API is capable of sending messages that do not fit entirely in RAM. To do so, the application must manage its output state and only write as many bytes as are available in the buffer at a time. The second `SMTPDemo` example provided in `main.c` sends a message that is a dump of all contents of the PIC's RAM. This example is currently commented out. Comment out the previous Short Message Example and uncomment the Long Message Example. This document will walk through sending a longer message.

Make sure `STACK_USE_SMTP_CLIENT` is uncommented in `tcPIP_config.h` before continuing.

Sending longer messages is divided into three stages. The first stage configures the SMTP client to send the message. The second stage sends the message in small chunks as buffer space is available. The final stage finishes the transmission and determines whether or not the message was successful.

The diagram below illustrates the first stage:

The first stage is largely similar to the first few steps in sending a short message. First, call `SMTPBeginUsage` to verify that the SMTP client is available and to begin a new message. If `FALSE` is returned, the SMTP client is busy and the application must return to the main loop to allow `StackTask` to execute again.

Next, set the local relay server to use as `SMTPClient.Server`. If the local relay server requires a user name and password, set `SMTPClient.Username` and `SMTPClient.Password` to the appropriate credentials.

If server parameters are not set, the stack will attempt to deliver the message directly to its destination host. This will likely fail due to spam prevention measures put in place by most ISPs and network administrators.

Continue on to set the header strings as necessary for the message. This includes the subject line, from address, and any recipients you need to add.

The next portion of the process differs. Ensure that `SMTPClient.Body` remains set to its default (NULL). At this point, call `SMTPSendMail` to open a connection to the remote server and transmit the headers. The application is now ready to proceed to the second stage and send the message body.

The following diagram provides an overview of stage two and three:

Upon entering stage two, the application should call `SMTPIsBusy` to verify that the connection to the remote server is active and has not been lost. If the call succeeds, call `SMTPIsPutReady` to determine how many bytes are available in the TX buffer. If no bytes are available, return to the main loop so that `StackTask` can transmit the data to the remote node and free up the buffer.

If space is available, any combination of the `SMTPPut`, `SMTPPutArray`, `SMTPPutROMArray`, `SMTPPutString`, and `SMTPPutROMString` functions may be called to transmit the message. These functions return the number of bytes successfully written. Use this value, along with the value originally returned from `SMTPIsPutReady` to track how much free space remains in the TX buffer. Once the buffer is depleted, call `SMTPFlush` to force the data written to be sent.

The SMTP client module can accept as much data as the TCP TX FIFO can hold. This is determined by the socket initializer for `TCP_PURPOSE_DEFAULT` type sockets in `tcPIP_config.h`, which defaults to 200 bytes.

If the TX buffer is exhausted before the message is complete, return to the main loop so that `StackTask` may transmit the data to the remote node and free up the buffer. Upon return, go to the beginning of the second stage to transmit the next portion of the message.

Once the message is complete, the application will move to the third stage. Call `SMTPPutDone` to inform the SMTP client

that no more data remains. Then call `SMTPIsBusy` repeatedly. Each time `TRUE` is returned, return to the main loop and wait for `StackTask` to execute again. Once `FALSE` is returned, the message transmission has completed and the application must call `SMTPEndUsage` to release the SMTP client. Check the return value of this function to determine if the message was successfully sent.

The example in `main.c` needs minor modifications to use your e-mail address. Set the `Server` and `To` fields in `SMTPDemo`, and ensure that these fields are being properly assigned to `SMTPClient` struct. The demo works exactly the same way as the previous one, with `BUTTON2` and `BUTTON3` held down simultaneously (the left-most two buttons) kicking off the state machine. LED1 will light as the message is being processed, and will extinguish when the SMTP state machine completes. If the transmission was successful LED2 will light, otherwise it will remain dark.

1.9.1.14.2 SMTP Client Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
≡	<code>SMTPBeginUsage</code>	
≡	<code>SMTPEndUsage</code>	This is function <code>SMTPEndUsage</code> .
≡	<code>SMTPFlush</code>	This is function <code>SMTPFlush</code> .
≡	<code>SMTPIsBusy</code>	This is function <code>SMTPIsBusy</code> .
≡	<code>SMTPIsPutReady</code>	This is function <code>SMTPIsPutReady</code> .
≡	<code>SMTPPutDone</code>	This is function <code>SMTPPutDone</code> .
≡	<code>SMTPSendMail</code>	This is function <code>SMTPSendMail</code> .

Macros

Name	Description
<code>SMTP_CONNECT_ERROR</code>	Connection to SMTP server failed
<code>SMTP_RESOLVE_ERROR</code>	DNS lookup for SMTP server failed
<code>SMTP_SUCCESS</code>	Message was successfully sent

Structures

Name	Description
<code>SMTP_POINTERS</code>	Configures the SMTP client to send a message

Variables

Name	Description
<code>SMTPClient</code>	

Description

The following functions and variables are available to the stack application.

1.9.1.14.2.1 SMTP_CONNECT_ERROR Macro

File

`smtp.h`

Syntax

```
#define SMTP_CONNECT_ERROR (0x8001u) // Connection to SMTP server failed
```

Description

Connection to SMTP server failed

1.9.1.14.2.2 SMTP_POINTERS Structure

Configures the SMTP client to send a message

File

smtp.h

Syntax

```
typedef struct {
    union {
        uint8_t * szRAM;
        ROM uint8_t * szROM;
    } Server;
    union {
        uint8_t * szRAM;
        ROM uint8_t * szROM;
    } Username;
    union {
        uint8_t * szRAM;
        ROM uint8_t * szROM;
    } Password;
    union {
        uint8_t * szRAM;
        ROM uint8_t * szROM;
    } To;
    union {
        uint8_t * szRAM;
        ROM uint8_t * szROM;
    } CC;
    union {
        uint8_t * szRAM;
        ROM uint8_t * szROM;
    } BCC;
    union {
        uint8_t * szRAM;
        ROM uint8_t * szROM;
    } From;
    union {
        uint8_t * szRAM;
        ROM uint8_t * szROM;
    } Subject;
    union {
        uint8_t * szRAM;
        ROM uint8_t * szROM;
    } OtherHeaders;
    union {
        uint8_t * szRAM;
        ROM uint8_t * szROM;
    } Body;
    struct {
        unsigned char Server : 1;
        unsigned char Username : 1;
        unsigned char Password : 1;
        unsigned char To : 1;
        unsigned char CC : 1;
        unsigned char BCC : 1;
        unsigned char From : 1;
        unsigned char Subject : 1;
        unsigned char OtherHeaders : 1;
        unsigned char Body : 1;
    } ROMPointers;
    bool UseSSL;
    uint16_t ServerPort;
} SMTP_POINTERS;
```

Description

This structure of pointers configures the SMTP Client to send an e-mail message. Initially, all pointers will be null. Set SMTPClient.[field name].szRAM to use a string stored in RAM, or SMTPClient.[field name].szROM to use a

string stored in ROM. (Where [field name] is one of the parameters below.)

If a ROM string is specified, `SMTPClient.ROMPointers.[field name]` must also be set to 1 to indicate that this field should be retrieved from ROM instead of RAM.

Remarks

When formatting an e-mail address, the SMTP standard format for associating a printable name may be used. This format places the printable name in quotation marks, with the address following in pointed brackets, such as "John Smith" <john.smith@domain.com>

Parameters

Parameters	Description
Server	the SMTP server to relay the message through
Username	the user name to use when logging into the SMTP server, if any is required
Password	the password to supply when logging in, if any is required
To	the destination address for this message. May be a comma-separated list of addresses, and/or formatted.
CC	The CC addresses for this message, if any. May be a comma-separated list of addresses, and/or formatted.
BCC	The BCC addresses for this message, if any. May be a comma-separated list of addresses, and/or formatted.
From	The From address for this message. May be formatted.
Subject	The Subject header for this message.
OtherHeaders	Any additional headers for this message. Each additional header, including the last one, must be terminated with a CRLF pair.
Body	When sending a message from memory, the location of the body of this message in memory. Leave as NULL to build a message on-the-fly.
ROMPointers	Indicates which parameters to read from ROM instead of RAM.
UseSSL	When <code>STACK_USE_SSL_CLIENT</code> is enabled, this flag causes the SMTP client to make an SSL connection to the server.
ServerPort	(uint16_t value) Indicates the port on which to connect to the remote SMTP server.

Function

```
typedef struct SMTP_POINTERS
```

1.9.1.14.2.3 SMTP_RESOLVE_ERROR Macro

File

smtp.h

Syntax

```
#define SMTP_RESOLVE_ERROR (0x8000u) // DNS lookup for SMTP server failed
```

Description

DNS lookup for SMTP server failed

1.9.1.14.2.4 SMTP_SUCCESS Macro

File

smtp.h

Syntax

```
#define SMTP_SUCCESS (0x0000u) // Message was successfully sent
```

Description

Message was successfully sent

1.9.1.14.2.5 SMTPBeginUsage Function**File**

smtp.h

Syntax

```
bool SMTPBeginUsage();
```

Section

SMTP Function Prototypes

1.9.1.14.2.6 SMTPClient Variable**File**

smtp.h

Syntax

```
SMTP_POINTERS SMTPClient;
```

Section

Global SMTP Variables

1.9.1.14.2.7 SMTPEndUsage Function**File**

smtp.h

Syntax

```
uint16_t SMTPEndUsage();
```

Description

This is function SMTPEndUsage.

1.9.1.14.2.8 SMTPFlush Function**File**

smtp.h

Syntax

```
void SMTPFlush();
```

Description

This is function SMTPFlush.

1.9.1.14.2.9 SMTPIsBusy Function**File**

smtp.h

Syntax

```
bool SMTPIsBusy();
```

Description

This is function SMTPIsBusy.

1.9.1.14.2.10 SMTPIsPutReady Function

File

smtp.h

Syntax

```
uint16_t SMTPIsPutReady();
```

Description

This is function SMTPIsPutReady.

1.9.1.14.2.11 SMTPPutDone Function

File

smtp.h

Syntax

```
void SMTPPutDone();
```

Description

This is function SMTPPutDone.

1.9.1.14.2.12 SMTPSendMail Function

File

smtp.h

Syntax

```
void SMTPSendMail();
```


Description

This is function SMTPSendMail.

1.9.1.14.3 SMTP Client Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
	SMTPTask	This is function SMTPTask.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.14.3.1 SMTPTask Function

File

smtp.h

Syntax

```
void SMTPTask( ) ;
```



Description

This is function SMTPTask.

1.9.1.14.4 SMTP Client Internal Members

Functions and variables used internally by the SMTP Client

Functions

	Name	Description
	SMTPPutROMArray	Non-ROM variant for C30 / C32
	SMTPPutROMString	Non-ROM variant for C30 / C32

Description

The following functions and variables are designated for internal use by the SMTP Client module.

1.9.1.14.4.1 SMTPPutROMArray Function

File

smtp.h

Syntax

```
uint16_t SMTPPutROMArray(ROM uint8_t * Data, uint16_t Len);
```

Description

Non-ROM variant for C30 / C32

1.9.1.14.4.2 SMTPPutROMString Function

File

smtp.h

Syntax

```
uint16_t SMTPPutROMString(ROM uint8_t * Data);
```

Description

Non-ROM variant for C30 / C32

1.9.1.15 Reboot

Provides a service to remotely reboot the PIC.

Description

The Reboot module will allow a user to remotely reboot the PIC microcontroller that is running the TCP/IP stack. This feature is primarily used for bootloader applications, which must reset the microcontroller to enter the bootloader code section. This module will execute a task that listens on a specified UDP port for a packet, and then reboots if it receives one. The port can be configured in `reboot.c` with the following macro:

```
#define REBOOT_PORT 69
```


For improved security, you can limit reboot capabilities to users on the same subnet by specifying the following macro in `reboot.c`:

```
#define REBOOT_SAME_SUBNET_ONLY
```


1.9.1.15.1 Reboot Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
	RebootTask	This is function RebootTask.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.15.1.1 RebootTask Function

File

reboot.h

Syntax

```
void RebootTask ( ) ;
```

Description

This is function RebootTask.

1.9.1.16 RSA

Performs public and private key encryption routines.

Description

The RSA module provides public and private key encryption support. This is used by the SSL module during the initial handshake in order to exchange a secret key for the remainder of the session.

The RSA module is covered by US Export Control law. It must be obtained separately from Microchip.



1.9.1.16.1 RSA Public Members

Functions and variables accessible by the stack application

Enumerations

Name	Description
RSA_DATA_FORMAT	Indicates the data format for any RSA integer
RSA_OP	Indicates the RSA operation to be completed
RSA_STATUS	Status response from RSA procedures

Functions

	Name	Description
	RSASendUsage	This is function RSASendUsage.
	RSASendStep	This is function RSASendStep.

Description

The following functions and variables are available to the stack application.

1.9.1.16.1.1 RSAEndUsage Function

File

rsa.h

Syntax

```
void RSAEndUsage();
```

Description

This is function RSAEndUsage.

1.9.1.16.1.2 RSAStep Function

File

rsa.h

Syntax

```
RSA_STATUS RSAStep();
```

Description

This is function RSAStep.

1.9.1.16.1.3 RSA_DATA_FORMAT Enumeration

File

rsa.h

Syntax

```
typedef enum {
    RSA_BIG_ENDIAN = 0u,
    RSA_LITTLE_ENDIAN
} RSA_DATA_FORMAT;
```

Members

Members	Description
RSA_BIG_ENDIAN = 0u	Data expressed with the most significant byte first
RSA_LITTLE_ENDIAN	Data expressed with the least significant byte first

Description

Indicates the data format for any RSA integer

1.9.1.16.1.4 RSA_OP Enumeration

File

rsa.h

Syntax

```
typedef enum {
    RSA_OP_ENCRYPT = 0u,
    RSA_OP_DECRYPT
} RSA_OP;
```

Members

Members	Description
RSA_OP_ENCRYPT = 0u	This is an encryption procedure
RSA_OP_DECRYPT	This is a decryption procedure

Description

Indicates the RSA operation to be completed

1.9.1.16.1.5 RSA_STATUS Enumeration**File**

rsa.h

Syntax

```
typedef enum {
    RSA_WORKING = 0u,
    RSA_FINISHED_M1,
    RSA_FINISHED_M2,
    RSA_DONE
} RSA_STATUS;
```

Members

Members	Description
RSA_WORKING = 0u	RSA is working through a calculation
RSA_FINISHED_M1	RSA decryption has just completed calculation of the M1 CRT value
RSA_FINISHED_M2	RSA decryption has just completed calculation of the M2 CRT value
RSA_DONE	The RSA calculation is complete

Description

Status response from RSA procedures

1.9.1.16.2 RSA Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
	RSASInit	

Macros

Name	Description
RSABeginDecrypt	This is macro RSABeginDecrypt.
RSABeginEncrypt	This is macro RSABeginEncrypt.
RSAEndDecrypt	This is macro RSAEndDecrypt.
RSAEndEncrypt	This is macro RSAEndEncrypt.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.16.2.1 RSABeginDecrypt Macro**File**

rsa.h

Syntax

```
#define RSABeginDecrypt RSABeginUsage(RSA_OP_DECRYPT, SSL_RSA_KEY_SIZE/8)
```

Description

This is macro RSABeginDecrypt.

1.9.1.16.2.2 RSABeginEncrypt Macro

File

rsa.h

Syntax

```
#define RSABeginEncrypt(a) RSABeginUsage(RSA_OP_ENCRYPT, a)
```

Description

This is macro RSABeginEncrypt.

1.9.1.16.2.3 RSAEndDecrypt Macro

File

rsa.h

Syntax

```
#define RSAEndDecrypt RSAEndUsage()
```

Description

This is macro RSAEndDecrypt.

1.9.1.16.2.4 RSAEndEncrypt Macro

File

rsa.h

Syntax

```
#define RSAEndEncrypt RSAEndUsage()
```

Description

This is macro RSAEndEncrypt.

1.9.1.16.2.5 RSAInit Function

File

rsa.h

Syntax

```
void RSAInit();
```

Section

Function Prototypes

1.9.1.16.3 RSA Internal Members

Functions and variables internal to the RSA module

Enumerations

Name	Description
SM_RSA	State machine for RSA processes

Macros

Name	Description
RSA_KEY_WORDS	Represents the number of words in a key
RSA_PRIME_WORDS	Represents the number of words in an RSA prime

Description

The following functions and variables are designated as internal to the RSA module.

1.9.1.16.3.1 RSA_KEY_WORDS Macro**File**

rsa.h

Syntax

```
#define RSA_KEY_WORDS (SSL_RSA_KEY_SIZE/BIGINT_DATA_SIZE) // Represents the number of words  
in a key
```

Description

Represents the number of words in a key

1.9.1.16.3.2 RSA_PRIME_WORDS Macro**File**

rsa.h

Syntax

```
#define RSA_PRIME_WORDS (SSL_RSA_KEY_SIZE/BIGINT_DATA_SIZE/2) // Represents the number of  
words in an RSA prime
```

Description

Represents the number of words in an RSA prime

1.9.1.16.3.3 SM_RSA Enumeration**File**

rsa.h

Syntax

```
typedef enum {  
    SM_RSA_IDLE = 0u,  
    SM_RSA_ENCRYPT_START,  
    SM_RSA_ENCRYPT,  
    SM_RSA_DECRYPT_START,  
    SM_RSA_DECRYPT_FIND_M1,  
    SM_RSA_DECRYPT_FIND_M2,  
    SM_RSA_DECRYPT_FINISH,  
    SM_RSA_DONE  
} SM_RSA;
```

Members

Members	Description
SM_RSA_IDLE = 0u	Data is being initialized by the application
SM_RSA_ENCRYPT_START	Initial state for encryption processes; encryption is ready to begin
SM_RSA_ENCRYPT	RSA encryption is proceeding
SM_RSA_DECRYPT_START	Initial state for decryption processes; decryption is ready to begin

SM_RSA_DECRYPT_FIND_M1	First stage in the CRT decryption algorithm
SM_RSA_DECRYPT_FIND_M2	Second stage in the CRT decryption algorithm
SM_RSA_DECRYPT_FINISH	CRT values have been calculated, so finalize the operation
SM_RSA_DONE	RSA process is complete

Description

State machine for RSA processes

1.9.1.17 SNTP Client

Obtains absolute time stamps from a pool of network time servers.

Description

The SNTP module implements the Simple Network Time Protocol. The module (by default) updates its internal time every 10 minutes using a pool of public global time servers. It then calculates reference times on any call to `SNTPGetUTCSeconds` using the internal Tick timer module.

The SNTP module is good for providing absolute time stamps. However, it should not be relied upon for measuring time differences (especially small differences). The pool of public time servers is implemented using round-robin DNS, so each update will come from a different server. Differing network delays and the fact that these servers are not verified implies that this time could be non-linear. While it is deemed reliable, it is not guaranteed to be accurate.


The Tick module provides much better accuracy (since it is driven by a hardware clock) and resolution, and should be used for measuring timeouts and other internal requirements.

Developers can change the value of `NTP_SERVER` if they wish to always point to a preferred time server, or to specify a region when accessing time servers. The default is to use the global pool.

1.9.1.17.1 SNTP Client Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
	<code>SNTPGetUTCSeconds</code>	This is function <code>SNTPGetUTCSeconds</code> .

Description

The following functions and variables are available to the stack application.

1.9.1.17.1.1 SNTPGetUTCSeconds Function

File

`sntp.h`

Syntax

```
uint32_t SNTPGetUTCSeconds();
```


Description

This is function `SNTPGetUTCSeconds`.

1.9.1.17.2 SNTP Client Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
	SNTPClient	This is function SNTPClient.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.17.2.1 SNTPClient Function**File**

sntp.h

Syntax

```
void SNTPClient();
```

Description

This is function SNTPClient.

1.9.1.17.3 SNTP Client Internal Members

Functions and variables internal to the SNTP Client module

Description

The following functions and variables are designated as internal to the SNTP Client module.

1.9.1.18 SSL

Implements SSL encryption for TCP connections.

Description

The SSL module adds encryption support to the TCP layer by implementing the SSLv3 protocol. This protocol is the standard for secure communications across the Internet, and prevents snooping or tampering of data as it travels across an untrusted network. This implementation of SSL supports the RSA asymmetric encryption protocol and the ARCFOUR symmetric encryption protocol.

Compiler	SSL Server Maximum RSA key length (SSL_RSA_KEY_SIZE)	SSL Client Maximum RSA key length (SSL_RSA_CLIENT_SIZE)
XC8	Not Supported	Not Supported
XC16	2048	2048
XC32	2048	2048

Previous versions of the MLA software distribution required that files containing cryptographic algorithm implementations be distributed separately to comply with U.S. Export Controls. In this version, the cryptographic modules are included with the TCP/IP Stack.

SSL Client Support

An SSL client can be initiated by first opening a TCP connection, then calling TCPStartSSLSession to initiate the SSL handshake process. The handshake uses the public key from the certificate provided by the server. Key lengths up to 1024 bits are supported on all processors; key lengths up to 2048 bits are supported on PIC32 microcontrollers. The `SSL_RSA_CLIENT_SIZE` macro in `ssl_client_size.h` sets the maximum certificate key length that your client should process.

```
#define SSL_RSA_CLIENT_SIZE      (2048ul)    // Size of Encryption Buffer (must be larger
than key size)
```

Once the handshake has started, call `TCPSSLIsHandshaking` until it returns `FALSE`. This will indicate that the handshake has completed and all traffic is now secured using 128-bit ARCFOUR encryption. If the handshake fails for any reason, the TCP connection will automatically be terminated as required by the SSL protocol specification.

For faster performance, the SSL module caches security parameters for the most recently made connections. This allows quick reconnections to the same node without the computational expense of another RSA handshake. By default, the two most recent connections are cached, but this can be modified in `tcpip_config.h`.

SSL client support is already enabled for SMTP. When `STACK_USE_SSL_CLIENT` is defined, the SMTP module automatically adds a field to `SMTPClient` called `UseSSL`. That field controls whether or not the SMTP client module will attempt to make an SSL connection before transmitting any data.

Note that TCP sockets using SSL may require an increase in TX/RX buffer size to support SSL. You can adjust the size of your sockets using the TCP/IP Configuration Utility included with the stack.

SSL Server Support

To initiate an SSL server, first open a TCP socket for listening using `TCPOpen`. Then call `TCPAddSSLListener` to listen for incoming SSL connections on an alternate port. This allows a single socket to share application-level resources and listen for connections on two different ports. Connections occurring on the originally opened port will proceed unsecured, while connections on the SSL port will first complete an SSL handshake to secure the data.

If your application will not accept unsecured traffic, simply open a non-secured socket on a free port number, then verify that each incoming connection is secured (not on that port) by calling `TCPIsSSL`.

SSL server support is automatically enabled for HTTP2 when `STACK_USE_SSL_SERVER` is defined. By default, the HTTP2 module will then listen for unsecured traffic on port 80 and secured connections on port 443.

This SSL server implementation supports key lengths up to 1024 bits on most PIC microcontrollers, and 2048 bits on PIC24 and PIC32 microcontrollers. The `SSL_RSA_KEY_SIZE` macro in `tcpip_config.h` sets the server certificate key length.

```
// Bits in SSL RSA key. This parameter is used for SSL sever
// connections only.
#define SSL_RSA_KEY_SIZE      (512ul)
```

Note that TCP sockets using SSL may require an increase in TX/RX buffer size to support SSL. You can adjust the size of your sockets using the TCP/IP Configuration Utility included with the stack.

Limitations

SSL was designed for desktop PCs with faster processors and significantly more resources than are available on an embedded platform. A few compromises must be made in order to use SSL in a less resource-intensive manner.

SSL Client Support

The SSL client module does not perform any validation or verification of certificates. Doing so would require many root certificates to be stored locally for verification, which is not feasible for memory-limited parts. This does not compromise security once the connection has been established, but does not provide complete security against man-in-the-middle attacks. (This sort of attack is uncommon and would be difficult to execute.)

Neither the SSL client nor the server can completely verify MACs before processing data. SSL records include a signature to verify that messages were not modified in transit. This Message Authentication Code, or MAC, is inserted after at least every 16kB of traffic. (It usually is inserted much more frequently than that.) Without 16kB of RAM to buffer packets for each socket, incoming data must be handed to the application layer before the MAC can be completely verified. Invalid MACs will still cause the connection to terminate immediately, but by the time this is detected some bad data may have already reached the application. Since the ARCFOUR cipher in use is a stream cipher, it would be difficult to exploit this in any meaningful way. An attacker would not be able to control what data is actually modified or inserted, as doing so without knowledge of the key would yield garbage. However, it is important to understand that incoming data is not completely verified before being passed to the application.

1.9.1.18.1 Generating Server Certificates

Describes how to import an SSL certificate into your code.

Description

The SSL certificates used by the TCP/IP Stack's SSL module are stored in the custom_ssl_cert.c source file. The following series of steps describe how to create the structures in custom_ssl_cert.c using an SSL certificate.

1. Download and install the OpenSSL library. There are several third-party sites that offer SSL installers (e.g. <http://www.slproweb.com/products/Win32OpenSSL.html>). Note that some distributions may not include all commands specified by the OpenSSL documentation.
2. Open a console and change directory to the **OpenSSL/bin** folder.
3. If you don't have a key and certificate, you can generate them first. The following example console commands will generate a 512-bit key:
 1. Generate the key: **openssl genrsa -out 512bits.key 512**
 2. Generate the Certificate Signing Request (CSR). You will need to add additional information when prompted: **openssl req -new -key 512bits.key -out 512bits.csr**
 3. Generate the X.509 certificate if self-signing (or send the CSR to a Certificate Authority for signing): **openssl x509 -req -days 365 -in 512bits.csr -signkey 512bits.key -out 512bits.crt** (note that if the -days option is not specified, the default expiration time is 30 days)
 4. For additional documentation, refer to <http://www.openssl.org/docs/apps/openssl.html>.

4. Parse your key file using the command: **openssl.exe asn1parse -in "[directory containing your key]\512bits.key"**

5. You should see a screen like this:

6. If you are not using an ENCX24J600 family device, then the last 5 integers displayed here are the SSL_P, SSL_Q, SSL_dP, SSL_dQ, and SSL_qInv parameters, respectively. However, they are displayed here in big-endian format, and the Microchip cryptographic library implementation requires parameters in little-endian format, so you will have to enter the parameters into the C arrays in opposite order. For example, the INTEGER at offset 145:

```
145:d=1 h1=2 := 33 prim: INTEGER
:D777566780029FCD610200B66D89507D
915E3E5BDB6FAB0233B5DFA2E4081DF7
```

will be swapped in the custom_ssl_cert.c file:

```
ROM BYTE SSL_P[] = {
    0xF7, 0x1D, 0x08, 0xE4, 0xA2, 0xDF, 0xB5, 0x33,
    0x02, 0xAB, 0x6F, 0xDB, 0x5B, 0x3E, 0x5E, 0x91,
    0x7D, 0x50, 0x89, 0x6D, 0xB6, 0x00, 0x02, 0x61,
    0xCD, 0x9F, 0x02, 0x80, 0x67, 0x56, 0x77, 0xD7
};
```

7. If you are using an ENCX24J600 family device, then the second and fourth integers displayed here are the SSL_N and SSL_D parameters, respectively. There is no need to do an endian format change for these parameters. For the example, the expected SSL_N and SSL_D values are shown in the figure below:

8. Parse your X.509 certificate using the command: **openssl.exe asn1parse -in "[directory containing your cert]\512bits.crt" -out cert.bin**

9. Open the cert.bin output file in a hex editor. For example, here is the default certificate information generated from 512bits.crt given in the stack:

10. This information must be copied verbatim into the SSL_CERT[] array. Note that this is binary data (not a large integer) so it does not get endian-swapped like the private key parameters.

```
ROM BYTE SSL_CERT[524] = {
    0x30, 0x82, 0x02, 0x08, 0x30, 0x82, 0x01, 0xb2, 0x02, 0x09, 0x00, 0xa5, 0x6a, 0xea, 0x1a, 0xa9,
    0x52, 0x9d, 0x1e, 0x30, 0x0d, 0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01, 0x05,
    0x05, 0x00, 0x30, 0x81, 0x8a, 0x31, 0x0b, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02,
    0x55, 0x53, 0x31, 0x10, 0x30, 0x0e, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x07, 0x41, 0x72, 0x69,
    0x7a, 0x6f, 0x6e, 0x61, 0x31, 0x11, 0x30, 0x0f, 0x06, 0x03, 0x55, 0x04, 0x07, 0x13, 0x08, 0x43,
    0x68, 0x61, 0x6e, 0x64, 0x6c, 0x65, 0x72, 0x31, 0x23, 0x30, 0x21, 0x06, 0x03, 0x55, 0x04, 0x0a,
    0x13, 0x1a, 0x4d, 0x69, 0x63, 0x72, 0x6f, 0x63, 0x68, 0x69, 0x70, 0x20, 0x54, 0x65, 0x63, 0x68,
```

```

0x6e, 0x6f, 0x6c, 0x6f, 0x67, 0x79, 0x2c, 0x20, 0x49, 0x6e, 0x63, 0x2e, 0x31, 0x1d, 0x30, 0x1b,
0x06, 0x03, 0x55, 0x04, 0x0b, 0x13, 0x14, 0x53, 0x53, 0x4c, 0x20, 0x44, 0x65, 0x6d, 0x6f, 0x20,
0x43, 0x65, 0x72, 0x74, 0x69, 0x66, 0x69, 0x63, 0x61, 0x74, 0x65, 0x31, 0x12, 0x30, 0x10, 0x06,
0x03, 0x55, 0x04, 0x03, 0x13, 0x09, 0x6d, 0x63, 0x68, 0x70, 0x62, 0x6f, 0x61, 0x72, 0x64, 0x30,
0x1e, 0x17, 0x0d, 0x30, 0x37, 0x31, 0x30, 0x30, 0x39, 0x31, 0x38, 0x33, 0x37, 0x32, 0x37, 0x5a,
0x17, 0x0d, 0x31, 0x37, 0x31, 0x30, 0x30, 0x36, 0x31, 0x38, 0x33, 0x37, 0x32, 0x37, 0x5a, 0x30,
0x81, 0x8a, 0x31, 0x0b, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x55, 0x53, 0x31,
0x10, 0x30, 0x0e, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x07, 0x41, 0x72, 0x69, 0x7a, 0x6f, 0x6e,
0x61, 0x31, 0x11, 0x30, 0x0f, 0x06, 0x03, 0x55, 0x04, 0x07, 0x13, 0x08, 0x43, 0x68, 0x61, 0x6e,
0x64, 0x6c, 0x65, 0x72, 0x31, 0x23, 0x30, 0x21, 0x06, 0x03, 0x55, 0x04, 0x0a, 0x13, 0x1a, 0x4d,
0x69, 0x63, 0x72, 0x6f, 0x63, 0x68, 0x69, 0x70, 0x20, 0x54, 0x65, 0x63, 0x68, 0x6e, 0x6f, 0x6c,
0x6f, 0x67, 0x79, 0x2c, 0x20, 0x49, 0x6e, 0x63, 0x2e, 0x31, 0x1d, 0x30, 0x1b, 0x06, 0x03, 0x55,
0x04, 0x0b, 0x13, 0x14, 0x53, 0x53, 0x4c, 0x20, 0x44, 0x65, 0x6d, 0x6f, 0x20, 0x43, 0x65, 0x72,
0x74, 0x69, 0x66, 0x69, 0x63, 0x61, 0x74, 0x65, 0x31, 0x12, 0x30, 0x10, 0x06, 0x03, 0x55, 0x04,
0x03, 0x13, 0x09, 0x6d, 0x63, 0x68, 0x70, 0x62, 0x6f, 0x61, 0x72, 0x64, 0x30, 0x5c, 0x30, 0x0d,
0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01, 0x01, 0x05, 0x00, 0x03, 0x4b, 0x00,
0x30, 0x48, 0x02, 0x41, 0x00, 0xaa, 0x96, 0xca, 0x97, 0xea, 0x27, 0xb0, 0xd7, 0xe9, 0x21, 0xd0,
0x40, 0xd4, 0x2c, 0x09, 0x5a, 0x2e, 0x3a, 0xe4, 0x12, 0x64, 0x2d, 0x4b, 0x1b, 0x92, 0xdf, 0x79,
0x68, 0x4e, 0x3c, 0x51, 0xf4, 0x43, 0x48, 0x0d, 0xf2, 0xc8, 0x50, 0x9b, 0x6e, 0xe5, 0xea, 0xfe,
0xef, 0xd9, 0x10, 0x41, 0x08, 0x14, 0xf9, 0x85, 0x49, 0xfc, 0x50, 0xd3, 0x57, 0x34, 0xdc, 0x3a,
0x0d, 0x79, 0xf8, 0xd3, 0x99, 0x02, 0x03, 0x01, 0x00, 0x01, 0x30, 0x0d, 0x06, 0x09, 0x2a, 0x86,
0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01, 0x05, 0x05, 0x00, 0x03, 0x41, 0x00, 0x18, 0x18, 0xfe, 0x8b,
0x2d, 0x0d, 0xf7, 0x0d, 0x65, 0x9d, 0x29, 0xec, 0xb3, 0x51, 0x6e, 0x3b, 0x93, 0xbb, 0x40, 0x1a,
0x0b, 0x34, 0x07, 0x63, 0x5e, 0x6a, 0x1c, 0x74, 0x59, 0xd4, 0x54, 0xd2, 0x1b, 0xf3, 0x31, 0xb7,
0x57, 0x4b, 0xa5, 0xe6, 0xe2, 0x35, 0xf7, 0xb3, 0x6a, 0x15, 0x6e, 0x3c, 0x93, 0x85, 0xb2, 0xca,
0xf5, 0x35, 0x00, 0xf4, 0x49, 0xe7, 0x00, 0x8a, 0x00, 0xd8, 0xe8, 0xcf
};

```

11. Update the `SSL_CERT_LEN` variable to contain the correct value.



1.9.1.18.2 SSL Public Members

Functions and variables accessible by the stack application

Enumerations

Name	Description
<code>SSL_SUPPLEMENTARY_DATA_TYPES</code>	This is type <code>SSL_SUPPLEMENTARY_DATA_TYPES</code> .

Functions

	Name	Description
	<code>TCPSSLIsHandshaking</code>	This is function <code>TCPSSLIsHandshaking</code> .
	<code>TCPIsSSL</code>	This is function <code>TCPIsSSL</code> .

Macros

Name	Description
<code>SSL_INVALID_ID</code>	Identifier for invalid SSL allocations
<code>SSL_RSA_CLIENT_SIZE</code>	Size of Encryption Buffer (must be larger than key size)

Structures

Name	Description
<code>SSL_PKEY_INFO</code>	To hash the public key information, we need to actually get the public key information... 1024 bit key at 8 bits/byte = 128 bytes needed for the public key.

Description

The following functions and variables are available to the stack application.

1.9.1.18.2.1 SSL_INVALID_ID Macro

File

ssl.h

Syntax

```
#define SSL_INVALID_ID (0xFFu) // Identifier for invalid SSL allocations
```

Description

Identifier for invalid SSL allocations

1.9.1.18.2.2 TCPSSLIsHandshaking Function

File

tcp.h

Syntax

```
bool TCPSSLIsHandshaking(TCP_SOCKET hTCP);
```

Description

This is function TCPSSLIsHandshaking.

1.9.1.18.2.3 TCPIsSSL Function

File

tcp.h

Syntax

```
bool TCPIsSSL(TCP_SOCKET hTCP);
```

Description

This is function TCPIsSSL.

1.9.1.18.2.4 SSL_SUPPLEMENTARY_DATA_TYPES Enumeration

File

ssl.h

Syntax

```
typedef enum {  
    SSL_SUPPLEMENTARY_DATA_NONE = 0,  
    SSL_SUPPLEMENTARY_DATA_CERT_PUBLIC_KEY  
} SSL_SUPPLEMENTARY_DATA_TYPES;
```

Description

This is type SSL_SUPPLEMENTARY_DATA_TYPES.

1.9.1.18.2.5 SSL_PKEY_INFO Structure

File

ssl.h

Syntax

```
typedef struct {  
    uint16_t pub_size_bytes;  
    uint8_t pub_key[SSL_RSA_CLIENT_SIZE / 8];  
    uint8_t pub_e[3];  
    uint8_t pub_guid;
```

```
} SSL_PKEY_INFO;
```

Members

Members	Description
uint8_t pub_guid;	This is used as a TCP_SOCKET which is a uint8_t

Description

To hash the public key information, we need to actually get the public key information... 1024 bit key at 8 bits/byte = 128 bytes needed for the public key.

1.9.1.18.2.6 SSL_RSA_CLIENT_SIZE Macro

File

ssl_client_size.h

Syntax

```
#define SSL_RSA_CLIENT_SIZE (2048ul) // Size of Encryption Buffer (must be larger than key size)
```

Description

Size of Encryption Buffer (must be larger than key size)

1.9.1.18.3 SSL Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
≡	SSLInit	
≡	TCPSSLGetPendingTxSize	This is function TCPSSLGetPendingTxSize.
≡	TCPSSLHandleIncoming	This is function TCPSSLHandleIncoming.
≡	TCPSSLHandshakeComplete	This is function TCPSSLHandshakeComplete.
≡	TCPStartSSLServer	This is function TCPStartSSLServer.

Macros

Name	Description
SSL_MIN_SESSION_LIFETIME	Minimum lifetime for SSL Sessions Sessions cannot be reallocated until this much time has elapsed
SSL_RSA_LIFETIME_EXTENSION	Lifetime extension for RSA operations Sessions lifetime is extended by this amount when an RSA calculation is made

Types

Name	Description
SSL_STATE	This is type SSL_STATE.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.18.3.1 SSL_STATE Type

File

tcp.h

Syntax

```
typedef enum SSL_STATE@1 SSL_STATE;
```

Description

This is type SSL_STATE.

1.9.1.18.3.2 SSLInit Function**File**

ssl.h

Syntax

```
void SSLInit();
```

Section

Function Prototypes

1.9.1.18.3.3 TCPSSLGetPendingTxSize Function**File**

tcp.h

Syntax

```
uint16_t TCPSSLGetPendingTxSize(TCP_SOCKET hTCP);
```

Description

This is function TCPSSLGetPendingTxSize.

1.9.1.18.3.4 TCPSSLHandleIncoming Function**File**

tcp.h

Syntax

```
void TCPSSLHandleIncoming(TCP_SOCKET hTCP);
```

Description

This is function TCPSSLHandleIncoming.

1.9.1.18.3.5 TCPSSLHandshakeComplete Function**File**

tcp.h

Syntax

```
void TCPSSLHandshakeComplete(TCP_SOCKET hTCP);
```

Description

This is function TCPSSLHandshakeComplete.

1.9.1.18.3.6 TCPStartSSLServer Function**File**

tcp.h

Syntax

```
bool TCPStartSSLServer(TCP_SOCKET hTCP);
```

Description

This is function TCPStartSSLServer.

1.9.1.18.3.7 SSL_MIN_SESSION_LIFETIME Macro**File**

ssl.h

Syntax

```
#define SSL_MIN_SESSION_LIFETIME (1*TICK_SECOND)
```

Description

Minimum lifetime for SSL Sessions Sessions cannot be reallocated until this much time has elapsed

1.9.1.18.3.8 SSL_RSA_LIFETIME_EXTENSION Macro**File**

ssl.h

Syntax

```
#define SSL_RSA_LIFETIME_EXTENSION (8*TICK_SECOND)
```

Description

Lifetime extension for RSA operations Sessions lifetime is extended by this amount when an RSA calculation is made


1.9.1.18.4 SSL Internal Members

Functions and variables internal to the SSL module

Enumerations

Name	Description
SM_SSL_RX_SERVER_HELLO	State machine for SSLRxServerHello
SSL_ALERT_LEVEL	Describes the two types of Alert records
SSL_MESSAGES	Describes the types of SSL messages (handshake and alerts)
SSL_SESSION_TYPE	SSL Session Type Enumeration

Functions

	Name	Description
	SSLRxHandshake	This is function SSLRxHandshake.

Macros

Name	Description
RESERVED_SSL_MEMORY	Total space needed by all SSL storage requirements
SSL_ALERT	Protocol code for Alert records
SSL_APPLICATION	Protocol code for Application data records
SSL_BUFFER_SIZE	Amount of space needed by a single SSL buffer
SSL_BUFFER_SPACE	Amount of space needed by all SSL buffer
SSL_CHANGE_CIPHER_SPEC	Protocol code for Change Cipher Spec records
SSL_HANDSHAKE	Protocol code for Handshake records
SSL_HASH_SIZE	Amount of space needed by a single SSL hash
SSL_HASH_SPACE	Amount of space needed by all SSL hash

SSL_KEYS_SIZE	Amount of space needed by a single SSL key
SSL_KEYS_SPACE	Amount of space needed by all SSL key
SSL_SESSION_SIZE	Amount of space needed by a single SSL session
SSL_SESSION_SPACE	Amount of space needed by all SSL session
SSL_STUB_SIZE	Amount of space needed by a single SSL stub
SSL_STUB_SPACE	Amount of space needed by all SSL stubs
SSL_VERSION	SSL version number
SSL_VERSION_HI	SSL version number (high byte)
SSL_VERSION_LO	SSL version number (low byte)
SSLFinishPartialRecord	This is macro SSLFinishPartialRecord.
SSLFlushPartialRecord	This is macro SSLFlushPartialRecord.

Structures

Name	Description
SSL_KEYS	Memory definition for SSL keys. This area is split into Local and Remote areas. During the handshake, Local.random and Remote.random hold the ServerRandom and ClientRandom values. Once the session keys are calculated, the Local.app and Remote.app contain the MAC secret, record sequence number, and encryption context for the ARCFOUR module.
SSL_SESSION	Storage space for SSL Session identifiers. (The SessionID and MasterSecret)
SSL_SESSION_STUB	Stub value for an SSL_SESSION. The tag associates this session with a remote node, either by matching to a remote IP address when we are the client or the first 3 bytes of the session ID when we are the host. When a session is free/expired, the tag is 0x00000000. The lastUsed value is the Tick count when the session was last used so that older sessions may be overwritten first.
SSL_STUB	Memory holder for general information associated with an SSL connections.

Unions

Name	Description
SSL_BUFFER	Generic buffer space for SSL. The hashRounds element is used when this buffer is needed for handshake hash calculations, and the full element is used as the Sbox for ARCFOUR calculations.

Description

The following functions and variables are designated as internal to the SSL module.

1.9.1.18.4.1 RESERVED_SSL_MEMORY Macro

File

ssl.h

Syntax

```
#define RESERVED_SSL_MEMORY ((uint32_t)(SSL_STUB_SPACE + SSL_KEYS_SPACE + SSL_HASH_SPACE + SSL_BUFFER_SPACE + SSL_SESSION_SPACE))
```

Description

Total space needed by all SSL storage requirements

1.9.1.18.4.2 SM_SSL_RX_SERVER_HELLO Enumeration

File

ssl.h

Syntax

```
typedef enum {
    RX_SERVER_CERT_START = 0u,
    RX_SERVER_CERT_FIND_KEY,
    RX_SERVER_CERT_FIND_N,
    RX_SERVER_CERT_READ_N,
    RX_SERVER_CERT_READ_E,
    RX_SERVER_CERT_CLEAR
} SM_SSL_RX_SERVER_HELLO;
```

Description

State machine for SSLRxServerHello

1.9.1.18.4.3 SSL_ALERT Macro

File

ssl.h

Syntax

```
#define SSL_ALERT 21u // Protocol code for Alert records
```

Description

Protocol code for Alert records

1.9.1.18.4.4 SSL_ALERT_LEVEL Enumeration

File

ssl.h

Syntax

```
typedef enum {
    SSL_ALERT_WARNING = 1u,
    SSL_ALERT_FATAL = 2u
} SSL_ALERT_LEVEL;
```

Members

Members	Description
SSL_ALERT_WARNING = 1u	Alert message is a warning (session can be resumed)
SSL_ALERT_FATAL = 2u	Alert message is fatal (session is non-resumable)

Description

Describes the two types of Alert records

1.9.1.18.4.5 SSL_APPLICATION Macro

File

ssl.h

Syntax

```
#define SSL_APPLICATION 23u // Protocol code for Application data records
```

Description

Protocol code for Application data records

1.9.1.18.4.6 SSL_BUFFER Union

File

ssl.h

Syntax

```
typedef union {
    struct {
        HASH_SUM hash;
        uint8_t md5_hash[16];
        uint8_t sha_hash[20];
        uint8_t temp[256 - sizeof (HASH_SUM) - 16 - 20];
    } hashRounds;
    uint8_t full[256];
} SSL_BUFFER;
```

Description

Generic buffer space for SSL. The hashRounds element is used when this buffer is needed for handshake hash calculations, and the full element is used as the Sbox for ARCFOUR calculations.

1.9.1.18.4.7 SSL_BUFFER_SIZE Macro**File**

ssl.h

Syntax

```
#define SSL_BUFFER_SIZE ((uint32_t)sizeof(SSL_BUFFER)) // Amount of space needed by a single SSL buffer
```

Description

Amount of space needed by a single SSL buffer

1.9.1.18.4.8 SSL_BUFFER_SPACE Macro**File**

ssl.h

Syntax

```
#define SSL_BUFFER_SPACE (SSL_BUFFER_SIZE*MAX_SSL_BUFFERS) // Amount of space needed by all SSL buffers
```

Description

Amount of space needed by all SSL buffers

1.9.1.18.4.9 SSL_CHANGE_CIPHER_SPEC Macro**File**

ssl.h

Syntax

```
#define SSL_CHANGE_CIPHER_SPEC 20u // Protocol code for Change Cipher Spec records
```

Description

Protocol code for Change Cipher Spec records

1.9.1.18.4.10 SSL_HANDSHAKE Macro**File**

ssl.h

Syntax

```
#define SSL_HANDSHAKE 22u // Protocol code for Handshake records
```

Description

Protocol code for Handshake records

1.9.1.18.4.11 SSL_HASH_SIZE Macro

File

ssl.h

Syntax

```
#define SSL_HASH_SIZE ((uint32_t)sizeof(HASH_SUM)) // Amount of space needed by a single SSL hash
```

Description

Amount of space needed by a single SSL hash

1.9.1.18.4.12 SSL_HASH_SPACE Macro

File

ssl.h

Syntax

```
#define SSL_HASH_SPACE ((uint32_t)(SSL_HASH_SIZE*MAX_SSL_HASHES)) // Amount of space needed by all SSL hash
```

Description

Amount of space needed by all SSL hash

1.9.1.18.4.13 SSL_KEYS Structure

File

ssl.h

Syntax

```
typedef struct {
    union {
        struct {
            uint8_t MACSecret[16];
            uint32_t sequence;
            ARCFOUR_CTX cryptCtx;
            uint8_t reserved[6];
        } app;
        uint8_t random[32];
    } Local;
    union {
        struct {
            uint8_t MACSecret[16];
            uint32_t sequence;
            ARCFOUR_CTX cryptCtx;
            uint8_t reserved[6];
        } app;
        uint8_t random[32];
    } Remote;
} SSL_KEYS;
```

Members

Members	Description
uint8_t MACSecret[16];	Server's MAC write secret
uint32_t sequence;	Server's write sequence number
ARCFOUR_CTX cryptCtx;	Server's write encryption context
uint8_t reserved[6];	Future expansion

uint8_t random[32];	Server.random value
uint8_t MACSecret[16];	Client's MAC write secret
uint32_t sequence;	Client's write sequence number
ARCFOUR_CTX cryptCtx;	Client's write encryption context
uint8_t reserved[6];	Future expansion
uint8_t random[32];	Client.random value

Description

Memory definition for SSL keys. This area is split into Local and Remote areas. During the handshake, Local.random and Remote.random hold the ServerRandom and ClientRandom values. Once the session keys are calculated, the Local.app and Remote.app contain the MAC secret, record sequence number, and encryption context for the ARCFOUR module.

1.9.1.18.4.14 SSL_KEYS_SIZE Macro**File**

ssl.h

Syntax

```
#define SSL_KEYS_SIZE ((uint32_t)sizeof(SSL_KEYS)) // Amount of space needed by a single
SSL key
```

Description

Amount of space needed by a single SSL key

1.9.1.18.4.15 SSL_KEYS_SPACE Macro**File**

ssl.h

Syntax

```
#define SSL_KEYS_SPACE (SSL_KEYS_SIZE*MAX_SSL_CONNECTIONS) // Amount of space needed by all
SSL key
```

Description

Amount of space needed by all SSL key

1.9.1.18.4.16 SSL_MESSAGES Enumeration**File**

ssl.h

Syntax

```
typedef enum {
    SSL_HELLO_REQUEST = 0u,
    SSL_CLIENT_HELLO = 1u,
    SSL_ANTIQUE_CLIENT_HELLO = 18u,
    SSL_SERVER_HELLO = 2u,
    SSL_CERTIFICATE = 11u,
    SSL_SERVER_HELLO_DONE = 14u,
    SSL_CLIENT_KEY_EXCHANGE = 16u,
    SSL_FINISHED = 20u,
    SSL_ALERT_CLOSE_NOTIFY = 0u+0x80,
    SSL_ALERT_UNEXPECTED_MESSAGE = 10u+0x80,
    SSL_ALERT_BAD_RECORD_MAC = 20u+0x80,
    SSL_ALERT_HANDSHAKE_FAILURE = 40u+0x80,
    SSL_NO_MESSAGE = 0xff
} SSL_MESSAGES;
```

Members

Members	Description
SSL_HELLO_REQUEST = 0u	HelloRequest handshake message (not currently supported)
SSL_CLIENT_HELLO = 1u	ClientHello handshake message
SSL_ANTIQUUE_CLIENT_HELLO = 18u	SSLv2 ClientHello handshake message (Supported for backwards compatibility. This is an internally defined value.)
SSL_SERVER_HELLO = 2u	ServerHello handshake message
SSL_CERTIFICATE = 11u	ServerCertificate handshake message
SSL_SERVER_HELLO_DONE = 14u	ServerHelloDone handshake message
SSL_CLIENT_KEY_EXCHANGE = 16u	ClientKeyExchange handshake message
SSL_FINISHED = 20u	Finished handshake message
SSL_ALERT_CLOSE_NOTIFY = 0u+0x80	CloseNotify alert message (dummy value used internally)
SSL_ALERT_UNEXPECTED_MESSAGE = 10u+0x80	UnexpectedMessage alert message (dummy value used internally)
SSL_ALERT_BAD_RECORD_MAC = 20u+0x80	BadRecordMAC alert message (dummy value used internally)
SSL_ALERT_HANDSHAKE_FAILURE = 40u+0x80	HandshakeFailure alert message (dummy value used internally)
SSL_NO_MESSAGE = 0xff	No message is currently requested (internally used value)

Description

Describes the types of SSL messages (handshake and alerts)

1.9.1.18.4.17 SSL_SESSION Structure**File**

ssl.h

Syntax

```
typedef struct {
    uint8_t sessionID[32];
    uint8_t masterSecret[48];
} SSL_SESSION;
```

Members

Members	Description
uint8_t sessionID[32];	The SSL Session ID for this session
uint8_t masterSecret[48];	Associated Master Secret for this session

Description

Storage space for SSL Session identifiers. (The SessionID and MasterSecret)

1.9.1.18.4.18 SSL_SESSION_SIZE Macro**File**

ssl.h

Syntax

```
#define SSL_SESSION_SIZE ((uint32_t)sizeof(SSL_SESSION)) // Amount of space needed by a single SSL session
```

Description

Amount of space needed by a single SSL session

1.9.1.18.4.19 SSL_SESSION_SPACE Macro

File

ssl.h

Syntax

```
#define SSL_SESSION_SPACE (SSL_SESSION_SIZE*MAX_SSL_SESSIONS) // Amount of space needed by all SSL session
```

Description

Amount of space needed by all SSL session

1.9.1.18.4.20 SSL_SESSION_STUB Structure

File

ssl.h

Syntax

```
typedef struct {  
    TCPIP_UINT32_VAL tag;  
    uint32_t lastUsed;  
} SSL_SESSION_STUB;
```

Members

Members	Description
TCPIP_UINT32_VAL tag;	Identifying tag for connection When we're a client, this is the remote IP When we're a host, this is 0x00 followed by first 3 bytes of session ID When this stub is free/expired, this is 0x00
uint32_t lastUsed;	Tick count when session was last used

Description

Stub value for an SSL_SESSION. The tag associates this session with a remote node, either by matching to a remote IP address when we are the client or the first 3 bytes of the session ID when we are the host. When a session is free/expired, the tag is 0x00000000. The lastUsed value is the Tick count when the session was last used so that older sessions may be overwritten first.

1.9.1.18.4.21 SSL_SESSION_TYPE Enumeration

File

ssl.h

Syntax

```
typedef enum {  
    SSL_CLIENT,  
    SSL_SERVER  
} SSL_SESSION_TYPE;
```

Members

Members	Description
SSL_CLIENT	Local device is the SSL client
SSL_SERVER	Local device is the SSL host

Description

SSL Session Type Enumeration

1.9.1.18.4.22 SSL_STUB Structure

File

ssl.h

Syntax

```
typedef struct {
    uint16_t wRxBytesRem;
    uint16_t wRxHsBytesRem;
    uint8_t rxProtocol;
    uint8_t rxHSType;
    uint8_t idSession;
    uint8_t idMD5, idSHA1;
    uint8_t idRxHash;
    uint8_t idRxBuffer, idTxBuffer;
    TCPIP_UINT32_VAL dwTemp;
    struct {
        unsigned char bIsServer : 1;
        unsigned char bClientHello : 1;
        unsigned char bServerHello : 1;
        unsigned char bServerCertificate : 1;
        unsigned char bServerHelloDone : 1;
        unsigned char bClientKeyExchange : 1;
        unsigned char bRemoteChangeCipherSpec : 1;
        unsigned char bRemoteFinished : 1;
        unsigned char bLocalChangeCipherSpec : 1;
        unsigned char bLocalFinished : 1;
        unsigned char bExpectingMAC : 1;
        unsigned char bNewSession : 1;
        unsigned char bCloseNotify : 1;
        unsigned char bDone : 1;
        unsigned char bRSAINProgress : 1;
        unsigned char bKeysValid : 1;
    } Flags;
    uint8_t requestedMessage;
    void * supplementaryBuffer;
    uint8_t supplementaryDataType;
} SSL_STUB;
```

Members

Members	Description
uint16_t wRxBytesRem;	Bytes left to read in current record
uint16_t wRxHsBytesRem;	Bytes left to read in current Handshake submessage
uint8_t rxProtocol;	Protocol for message being read
uint8_t rxHSType;	Handshake message being received
uint8_t idSession;	ID for associated session
uint8_t idRxHash;	ID for MAC hash (TX needs no persistence)
TCPIP_UINT32_VAL dwTemp;	Used for state machine in RxCertificate
unsigned char bIsServer : 1;	We are the server
unsigned char bClientHello : 1;	ClientHello has been sent/received
unsigned char bServerHello : 1;	ServerHello has been sent/received
unsigned char bServerCertificate : 1;	ServerCertificate has been sent/received
unsigned char bServerHelloDone : 1;	ServerHelloDone has been sent/received
unsigned char bClientKeyExchange : 1;	ClientKeyExchange has been sent/received
unsigned char bRemoteChangeCipherSpec : 1;	Remote node has sent a ChangeCipherSpec message
unsigned char bRemoteFinished : 1;	Remote node has sent a Finished message
unsigned char bLocalChangeCipherSpec : 1;	We have sent a ChangeCipherSpec message
unsigned char bLocalFinished : 1;	We have sent a Finished message
unsigned char bExpectingMAC : 1;	We expect a MAC at end of message
unsigned char bNewSession : 1;	true if a new session, false if resuming

unsigned char bCloseNotify : 1;	Whether or not a CloseNotify has been sent/received
unsigned char bDone : 1;	true if the connection is closed
unsigned char bRSAInProgress : 1;	true when RSA op is in progress
unsigned char bKeysValid : 1;	true if the session keys have been generated
uint8_t requestedMessage;	Currently requested message to send, or 0xff

Description

Memory holder for general information associated with an SSL connections.

1.9.1.18.4.23 SSL_STUB_SIZE Macro**File**

ssl.h

Syntax

```
#define SSL_STUB_SIZE ((uint32_t)sizeof(SSL_STUB)) // Amount of space needed by a single SSL stub
```

Description

Amount of space needed by a single SSL stub

1.9.1.18.4.24 SSL_STUB_SPACE Macro**File**

ssl.h

Syntax

```
#define SSL_STUB_SPACE (SSL_STUB_SIZE*MAX_SSL_CONNECTIONS) // Amount of space needed by all SSL stubs
```

Description

Amount of space needed by all SSL stubs

1.9.1.18.4.25 SSL_VERSION Macro**File**

ssl.h

Syntax

```
#define SSL_VERSION (0x0300u) // SSL version number
```

Description

SSL version number

1.9.1.18.4.26 SSL_VERSION_HI Macro**File**

ssl.h

Syntax

```
#define SSL_VERSION_HI (0x03u) // SSL version number (high byte)
```

Description

SSL version number (high byte)

1.9.1.18.4.27 SSL_VERSION_LO Macro

File

ssl.h

Syntax

```
#define SSL_VERSION_LO (0x00u) // SSL version number (low byte)
```

Description

SSL version number (low byte)

1.9.1.18.4.28 SSLFinishPartialRecord Macro

File

ssl.h

Syntax

```
#define SSLFinishPartialRecord(a) TCPSSLPutRecordHeader(a, NULL, true);
```

Description

This is macro SSLFinishPartialRecord.

1.9.1.18.4.29 SSLFlushPartialRecord Macro

File

ssl.h

Syntax

```
#define SSLFlushPartialRecord(a) TCPSSLPutRecordHeader(a, NULL, false);
```

Description

This is macro SSLFlushPartialRecord.

1.9.1.18.4.30 SSLRxHandshake Function

File

ssl.h

Syntax

```
void SSLRxHandshake(TCP_SOCKET hTCP);
```

Description

This is function SSLRxHandshake.

1.9.1.19 TCP

Implements the TCP transport layer protocol.

Description

TCP is a standard transport layer protocol described in RFC 793. It provides reliable stream-based connections over unreliable networks, and forms the foundation for HTTP, SMTP, and many other protocol standards.

Connections made over TCP guarantee data transfer at the expense of throughput. Connections are made through a three-way handshake process, ensuring a one-to-one connection. Remote nodes advertise how much data they are ready to receive, and all data transmitted must be acknowledged. If a remote node fails to acknowledge the receipt of data, it is automatically retransmitted. This ensures that network errors such as lost, corrupted, or out-of-order packets are

automatically corrected.

To accomplish this, TCP must operate in a buffer. Once the transmit buffer is full, no more data can be sent until the remote node has acknowledged receipt. For the Microchip TCP/IP Stack, the application must return to the main stack loop in order for this to happen. Likewise, the remote node cannot transmit more data until the local device has acknowledged receipt and that space is available in the buffer. When a local application needs to read more data, it must return to the main stack loop and wait for a new packet to arrive.

The TCP flow diagram below provides an overview for the use of the TCP module:

Sockets are opened using TCPOpen. This function can either open a listening socket to wait for client connections, or can make a client connection to the remote node. The remote node can be specified by a host name string to be resolved in DNS, an IP address, or a NODE_INFO struct containing previously resolved IP and MAC address information.

Once connected, applications can read and write data. On each entry, the application must verify that the socket is still connected. For most applications a call to TCPIsConnected will be sufficient, but TCPWasReset may also be used for listening sockets that may turn over quickly.

To write data, call TCPIsPutReady to check how much space is available. Then, call any of the TCPput family of functions to write data as space is available. Once complete, call TCPFlush to transmit data immediately. Alternately, return to the main stack loop. Data will be transmitted when either a) half of the transmit buffer becomes full or b) a delay time has passed (usually 40ms).

To read data, call TCPIsGetReady to determine how many bytes are ready to be retrieved. Then use the TCPGet family of functions to read data from the socket, and/or the TCPFind family of functions to locate data in the buffer. When no more data remains, return to the main stack loop to wait for more data to arrive.

If the application needs to close the connection, call TCPDisconnect, then return to the main stack loop and wait for the remote node to acknowledge the disconnection. Client sockets will return to the idle state, while listening sockets will wait for a new connection.

For more information, refer to the GenericTCPClient or GenericTCPServer examples, or read the associated RFC.

1.9.1.19.1 TCP Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
≡	TCPClose	This is function TCPClose.
≡	TCPDiscard	This is function TCPDiscard.
≡	TCPDisconnect	This is function TCPDisconnect.
≡	TCPFlush	This is function TCPFlush.
≡	TCPGetRemoteInfo	This is function TCPGetRemoteInfo.
≡	TCPGetRxFIFOFree	This is function TCPGetRxFIFOFree.
≡	TCPGetTxFIFOFull	This is function TCPGetTxFIFOFull.
≡	TCPIsConnected	This is function TCPIsConnected.
≡	TCPIsGetReady	This is function TCPIsGetReady.
≡	TCPIsPutReady	This is function TCPIsPutReady.
≡	TCPWasReset	This is function TCPWasReset.

Macros

Name	Description
INVALID_SOCKET	The socket is invalid or could not be opened
UNKNOWN_SOCKET	The socket is not known
TCP_ADJUST_GIVE_REST_TO_RX	Resize flag: extra bytes go to RX
TCP_ADJUST_GIVE_REST_TO_TX	Resize flag: extra bytes go to TX

TCP_ADJUST_PRESERVE_RX	Resize flag: attempt to preserve RX buffer
TCP_ADJUST_PRESERVE_TX	Resize flag: attempt to preserve TX buffer
TCP_OPEN_IP_ADDRESS	Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_IP_ADDRESS while STACK_CLIENT_MODE feature is not enabled.
TCP_OPEN_NODE_INFO	Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_NODE_INFO while STACK_CLIENT_MODE feature is not enabled.
TCP_OPEN_RAM_HOST	Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_RAM_HOST while STACK_CLIENT_MODE feature is not enabled.
TCP_OPEN_ROM_HOST	Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_ROM_HOST while STACK_CLIENT_MODE feature is not enabled.
TCP_OPEN_SERVER	Create a server socket and ignore dwRemoteHost.
TCPConnect	Alias to TCPOpen as a client.
TCPFind	Alias to TCPFindEx with no length parameter.
TCPFindArray	Alias to TCPFindArrayEx with no length parameter.
TCPFindROMArray	Alias to TCPFindROMArrayEx with no length parameter.
TCPFindROMArrayEx	This is macro TCPFindROMArrayEx.
TCPGetRxFIFOFull	Alias to TCPIsGetReady provided for API completeness
TCPGetTxFIFOFree	Alias to TCPIsPutReady provided for API completeness
TCPListen	Alias to TCPOpen as a server.
TCPPutROMArray	This is macro TCPPutROMArray.
TCPPutROMString	This is macro TCPPutROMString.

Description

The following functions and variables are available to the stack application.

1.9.1.19.1.1 INVALID_SOCKET Macro**File**

tcp.h

Syntax

```
#define INVALID_SOCKET (0xFE) // The socket is invalid or could not be opened
```

Description

The socket is invalid or could not be opened

1.9.1.19.1.2 UNKNOWN_SOCKET Macro**File**

tcp.h

Syntax

```
#define UNKNOWN_SOCKET (0xFF) // The socket is not known
```

Description

The socket is not known

1.9.1.19.1.3 TCP_ADJUST_GIVE_REST_TO_RX Macro**File**

tcp.h

Syntax

```
#define TCP_ADJUST_GIVE_REST_TO_RX 0x01u // Resize flag: extra bytes go to RX
```

Description

Resize flag: extra bytes go to RX

1.9.1.19.1.4 TCP_ADJUST_GIVE_REST_TO_TX Macro**File**

tcp.h

Syntax

```
#define TCP_ADJUST_GIVE_REST_TO_TX 0x02u // Resize flag: extra bytes go to TX
```

Description

Resize flag: extra bytes go to TX

1.9.1.19.1.5 TCP_ADJUST_PRESERVE_RX Macro**File**

tcp.h

Syntax

```
#define TCP_ADJUST_PRESERVE_RX 0x04u // Resize flag: attempt to preserve RX buffer
```

Description

Resize flag: attempt to preserve RX buffer

1.9.1.19.1.6 TCP_ADJUST_PRESERVE_TX Macro**File**

tcp.h

Syntax

```
#define TCP_ADJUST_PRESERVE_TX 0x08u // Resize flag: attempt to preserve TX buffer
```

Description

Resize flag: attempt to preserve TX buffer

1.9.1.19.1.7 TCP_OPEN_IP_ADDRESS Macro**File**

tcp.h

Syntax

```
#define TCP_OPEN_IP_ADDRESS You_need_to_enable_STACK_CLIENT_MODE_to_use_TCP_OPEN_IP_ADDRESS
```

Description

Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_IP_ADDRESS while STACK_CLIENT_MODE feature is not enabled.

1.9.1.19.1.8 TCP_OPEN_NODE_INFO Macro**File**

tcp.h

Syntax

```
#define TCP_OPEN_NODE_INFO You_need_to_enable_STACK_CLIENT_MODE_to_use_TCP_OPEN_NODE_INFO
```

Description

Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_NODE_INFO while STACK_CLIENT_MODE feature is not enabled.

1.9.1.19.1.9 TCP_OPEN_RAM_HOST Macro**File**

tcp.h

Syntax

```
#define TCP_OPEN_RAM_HOST You_need_to_enable_STACK_CLIENT_MODE_to_use_TCP_OPEN_RAM_HOST
```

Description

Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_RAM_HOST while STACK_CLIENT_MODE feature is not enabled.

1.9.1.19.1.10 TCP_OPEN_ROM_HOST Macro**File**

tcp.h

Syntax

```
#define TCP_OPEN_ROM_HOST You_need_to_enable_STACK_CLIENT_MODE_to_use_TCP_OPEN_ROM_HOST
```

Description

Emit an undeclared identifier diagnostic if code tries to use TCP_OPEN_ROM_HOST while STACK_CLIENT_MODE feature is not enabled.

1.9.1.19.1.11 TCP_OPEN_SERVER Macro**File**

tcp.h

Syntax

```
#define TCP_OPEN_SERVER 0u
```

Description

Create a server socket and ignore dwRemoteHost.

1.9.1.19.1.12 TCPConnect Macro

Alias to TCPOpen as a client.

File

tcp.h

Syntax

```
#define TCPConnect(remote,port) TCPOpen((uint32_t)remote, TCP_OPEN_NODE_INFO, port, TCP_PURPOSE_DEFAULT)
```

Description

This function is an alias to TCPOpen for client sockets. It is provided for backwards compatibility with older versions of the stack. New applications should use the TCPOpen API instead.

1.9.1.19.1.13 TCPClose Function

File

tcp.h

Syntax

```
void TCPClose(TCP_SOCKET hTCP);
```

Description

This is function TCPClose.

1.9.1.19.1.14 TCPDiscard Function

File

tcp.h

Syntax

```
void TCPDiscard(TCP_SOCKET hTCP);
```

Description

This is function TCPDiscard.

1.9.1.19.1.15 TCPDisconnect Function

File

tcp.h

Syntax

```
void TCPDisconnect(TCP_SOCKET hTCP);
```

Description

This is function TCPDisconnect.

1.9.1.19.1.16 TCPFind Macro

Alias to TCPFindEx with no length parameter.

File

tcp.h

Syntax

```
#define TCPFind(a,b,c,d) TCPFindEx(a,b,c,0,d)
```

Description

This function is an alias to TCPFindEx with no length parameter. It is provided for backwards compatibility with an older API.

1.9.1.19.1.17 TCPFindArray Macro

Alias to TCPFindArrayEx with no length parameter.

File

tcp.h

Syntax

```
#define TCPFindArray(a,b,c,d,e) TCPFindArrayEx(a,b,c,d,0,e)
```

Description

This function is an alias to TCPFindArrayEx with no length parameter. It is provided for backwards compatibility with an older

API.

1.9.1.19.1.18 TCPFindROMArray Macro

Alias to TCPFindROMArrayEx with no length parameter.

File

tcp.h

Syntax

```
#define TCPFindROMArray(a,b,c,d,e) TCPFindArray(a,(uint8_t *)b,c,d,e)
```

Description

This function is an alias to TCPFindROMArrayEx with no length parameter. It is provided for backwards compatibility with an older API.

1.9.1.19.1.19 TCPFindROMArrayEx Macro

File

tcp.h

Syntax

```
#define TCPFindROMArrayEx(a,b,c,d,e,f) TCPFindArrayEx(a,(uint8_t *)b,c,d,e,f)
```

Description

This is macro TCPFindROMArrayEx.

1.9.1.19.1.20 TCPFlush Function

File

tcp.h

Syntax

```
void TCPFlush(TCP_SOCKET hTCP);
```

Description

This is function TCPFlush.

1.9.1.19.1.21 TCPGetRemoteInfo Function

File

tcp.h

Syntax

```
SOCKET_INFO * TCPGetRemoteInfo(TCP_SOCKET hTCP);
```

Description

This is function TCPGetRemoteInfo.

1.9.1.19.1.22 TCPGetRxFIFOFree Function

File

tcp.h

Syntax

```
uint16_t TCPGetRxFIFOFree(TCP_SOCKET hTCP);
```

Description

This is function TCPGetRxFIFOFree.

1.9.1.19.1.23 TCPGetRxFIFOFull Macro**File**

tcp.h

Syntax

```
#define TCPGetRxFIFOFull(a) TCPIsGetReady(a)
```

Description

Alias to TCPIsGetReady provided for API completeness

1.9.1.19.1.24 TCPGetTxFIFOFree Macro**File**

tcp.h

Syntax

```
#define TCPGetTxFIFOFree(a) TCPIsPutReady(a)
```

Description

Alias to TCPIsPutReady provided for API completeness

1.9.1.19.1.25 TCPGetTxFIFOFull Function**File**

tcp.h

Syntax

```
uint16_t TCPGetTxFIFOFull(TCP_SOCKET hTCP);
```

Description

This is function TCPGetTxFIFOFull.

1.9.1.19.1.26 TCPIsConnected Function**File**

tcp.h

Syntax

```
bool TCPIsConnected(TCP_SOCKET hTCP);
```

Description

This is function TCPIsConnected.

1.9.1.19.1.27 TCPIsGetReady Function**File**

tcp.h

Syntax

```
uint16_t TCPIsGetReady(TCP_SOCKET hTCP);
```

Description

This is function TCPIsGetReady.

1.9.1.19.1.28 TCPIsPutReady Function**File**

tcp.h

Syntax

```
uint16_t TCPIsPutReady(TCP_SOCKET hTCP);
```

Description

This is function TCPIsPutReady.

1.9.1.19.1.29 TCPListen Macro

Alias to TCPOpen as a server.

File

tcp.h

Syntax

```
#define TCPListen(port) TCPOpen(0, TCP_OPEN_SERVER, port, TCP_PURPOSE_DEFAULT)
```

Description

This function is an alias to TCPOpen for server sockets. It is provided for backwards compatibility with older versions of the stack. New applications should use the TCPOpen API instead.

1.9.1.19.1.30 TCPPutROMArray Macro**File**

tcp.h

Syntax

```
#define TCPPutROMArray(a,b,c) TCPPutArray(a,(uint8_t *)b,c)
```

Description

This is macro TCPPutROMArray.

1.9.1.19.1.31 TCPPutROMString Macro**File**

tcp.h

Syntax

```
#define TCPPutROMString(a,b) TCPPutString(a,(uint8_t *)b)
```

Description

This is macro TCPPutROMString.

1.9.1.19.1.32 TCPWasReset Function**File**

tcp.h

Syntax

```
bool TCPWasReset(TCP_SOCKET hTCP);
```




Description

This is function TCPWasReset.

1.9.1.19.2 TCP Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
	TCPIInit	
	TCPTick	This is function TCPTick.

Types

Name	Description
SOCKET_INFO	Information about a socket
TCB	Remainder of TCP Control Block data. The rest of the TCB is stored in Ethernet buffer RAM or elsewhere as defined by vMemoryMedium. Current size is 41 (PIC18), 42 (PIC24), or 48 bytes (PIC32)
TCB_STUB	TCP Control Block (TCB) stub data storage. Stubs are stored in local PIC RAM for speed. Current size is 34 bytes (PIC18), 36 bytes (PIC24), or 56 (PIC32)
TCP_SOCKET	A TCP_SOCKET is stored as a single uint8_t
TCP_STATE	TCP States as defined by RFC 793

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.19.2.1 SOCKET_INFO Type

File

tcp.h

Syntax

```
typedef struct SOCKET_INFO@1 SOCKET_INFO;
```

Description

Information about a socket

1.9.1.19.2.2 TCB Type

File

tcp.h

Syntax

```
typedef struct TCB@1 TCB;
```

Description

Remainder of TCP Control Block data. The rest of the TCB is stored in Ethernet buffer RAM or elsewhere as defined by vMemoryMedium. Current size is 41 (PIC18), 42 (PIC24), or 48 bytes (PIC32)

1.9.1.19.2.3 TCB_STUB Type

File

tcp.h

Syntax

```
typedef struct TCB_STUB@1 TCB_STUB;
```

Description

TCP Control Block (TCB) stub data storage. Stubs are stored in local PIC RAM for speed. Current size is 34 bytes (PIC18), 36 bytes (PIC24), or 56 (PIC32)

1.9.1.19.2.4 TCP_SOCKET Type**File**

tcp.h

Syntax

```
typedef uint8_t TCP_SOCKET;
```

Description

A TCP_SOCKET is stored as a single uint8_t

1.9.1.19.2.5 TCP_STATE Type**File**

tcp.h

Syntax

```
typedef enum TCP_STATE@1 TCP_STATE;
```

Description

TCP States as defined by RFC 793

1.9.1.19.2.6 TCPInit Function**File**

tcp.h

Syntax

```
void TCPInit();
```

Section

Function Declarations

1.9.1.19.2.7 TCPTick Function**File**

tcp.h

Syntax

```
void TCPTick();
```

Description

This is function TCPTick.

1.9.1.19.3 TCP Internal Members

Functions and variables internal to the TCP module

Description

The following functions and variables are designated as internal to the TCP module.

1.9.1.20 Telnet

Describes the operation of the Telnet module.

Description

Telnet provides bidirectional, interactive communication between two nodes on the Internet or on a Local Area Network. The Telnet code included with Microchip's TCP/IP stack is a demonstration of the structure of a Telnet application. This demo begins by listening for a Telnet connection. When a client attempts to make one, the demo will prompt the client for a username and password, and if the correct one is provided, will output and periodically refresh several values obtained from the demo board.

There are several changes that you may need to make to `telnet.c` and/or `telnet.h` to suit your application. All of the Telnet Public members can be re-defined in the application-specific section of `tcPIP_config.h`. You may also wish to change some of the Telnet Internal Member strings, located in `telnet.c`, to more accurately reflect your application. You will also need to modify the `TelnetTask` function to include the functionality you'd like. You may insert or change states in `TelnetTask` as needed.

1.9.1.20.1 Telnet Public Members

Functions and variables accessible by the stack application


Description

The following functions and variables are available to the stack application.

1.9.1.20.2 Telnet Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
	TelnetTask	This is function TelnetTask.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.20.2.1 TelnetTask Function

File

telnet.h

Syntax

```
void TelnetTask();
```

Description

This is function TelnetTask.

1.9.1.20.3 Telnet Internal Members

Functions and variables internal to the Telnet module

Description

The following functions and variables are designated as internal to the Telnet module.

1.9.1.21 TFTP

Describes the TFTP module.




Description

The Trivial File Transfer Protocol provides unreliable upload and download services to applications connected to the UDP-based TFTP server.










1.9.1.21.1 TFTP Public Members

Functions and variables accessible by the stack application.

Enumerations

	Name	Description
	TFTP_ACCESS_ERROR	Standard error codes as defined by TFTP spec. Use to decode value returned by TFTPGetError().
	_TFTP_ACCESS_ERROR	Standard error codes as defined by TFTP spec. Use to decode value returned by TFTPGetError().
	TFTP_FILE_MODE	File open mode as used by TFTPFileOpen().
	_TFTP_FILE_MODE	File open mode as used by TFTPFileOpen().
	TFTP_RESULT	Enum. of results returned by most of the TFTP functions.
	_TFTP_RESULT	Enum. of results returned by most of the TFTP functions.

Functions

	Name	Description
	TFTPCloseFile	This is function TFTPCloseFile.
	TFTPGet	This is function TFTPGet.
	TFTPIsFileClosed	This is function TFTPIsFileClosed.
	TFTPIsFileOpened	This is function TFTPIsFileOpened.
	TFTPIsGetReady	This is function TFTPIsGetReady.
	TFTPIsOpened	This is function TFTPIsOpened.
	TFTPIsPutReady	This is function TFTPIsPutReady.
	TFTPOpen	This is function TFTPOpen.
	TFTPGetUploadStatus	This is function TFTPGetUploadStatus.

Macros

Name	Description
TFTPClose	Macro: void TFTPClose(void) Closes TFTP client socket.
TFTPGetError	Macro: uint16_t TFTPGetError(void) Returns previously saved error code.
TFTPIsFileOpenReady	Macro: bool TFTPIsFileOpenReady(void) Checks to see if it is okay to send TFTP file open request to remote server.
TFTPOpenROMFile	This is macro TFTPOpenROMFile.
TFTP_UPLOAD_COMPLETE	Status codes for TFTPGetUploadStatus() function. Zero means upload success, >0 means working and <0 means fatal error.
TFTP_UPLOAD_CONNECT	This is macro TFTP_UPLOAD_CONNECT.

TFTP_UPLOAD_CONNECT_TIMEOUT	This is macro TFTP_UPLOAD_CONNECT_TIMEOUT.
TFTP_UPLOAD_GET_DNS	This is macro TFTP_UPLOAD_GET_DNS.
TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT	This is macro TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT.
TFTP_UPLOAD_RESOLVE_HOST	This is macro TFTP_UPLOAD_RESOLVE_HOST.
TFTP_UPLOAD_SEND_DATA	This is macro TFTP_UPLOAD_SEND_DATA.
TFTP_UPLOAD_SEND_FILENAME	This is macro TFTP_UPLOAD_SEND_FILENAME.
TFTP_UPLOAD_SERVER_ERROR	This is macro TFTP_UPLOAD_SERVER_ERROR.
TFTP_UPLOAD_WAIT_FOR_CLOSURE	This is macro TFTP_UPLOAD_WAIT_FOR_CLOSURE.

Types

Name	Description
TFTP_CHUNK_DESCRIPTOR	This is type TFTP_CHUNK_DESCRIPTOR.

Description

The following functions and variables are available to the stack application.

1.9.1.21.1.1 TFTPclose Macro**File**

tftp.h

Syntax

```
#define TFTPclose(void) UDPClose(_tftpSocket)
```

Side Effects

None

Returns

None

Description

Macro: void TFTPclose(void)

Closes TFTP client socket.

Remarks

Once closed, application must do TFTPOpen to perform any new TFTP operations.

If TFTP server does not change during application life-time, one may not need to call TFTPclose and keep TFTP socket open.

Preconditions

TFTPOpen is already called and TFTPisOpened() returned TFTP_OK.

1.9.1.21.1.2 TFTPcloseFile Function**File**

tftp.h

Syntax

```
void TFTPcloseFile();
```

Description

This is function TFTPcloseFile.

1.9.1.21.1.3 TFTPGet Function

File

tftp.h

Syntax

```
uint8_t TFTPGet();
```

Description

This is function TFTPGet.

1.9.1.21.1.4 TFTPGetError Macro

File

tftp.h

Syntax

```
#define TFTPGetError (_tftpError)
```

Side Effects

None

Returns

Error code as returned by remote server. Application may use TFTP_ACCESS_ERROR enum. to decode standard error code.

Description

Macro: uint16_t TFTPGetError(void)

Returns previously saved error code.

Remarks

None

Preconditions

One of the TFTP function returned with TFTP_ERROR result.

1.9.1.21.1.5 TFTPIsFileClosed Function

File

tftp.h

Syntax

```
TFTP_RESULT TFTPIsFileClosed();
```

Description

This is function TFTPIsFileClosed.

1.9.1.21.1.6 TFTPIsFileOpened Function

File

tftp.h

Syntax

```
TFTP_RESULT TFTPIsFileOpened();
```

Description

This is function TFTPIsFileOpened.

1.9.1.21.1.7 TFTP_IsFileOpenReady Macro

File

tftp.h

Syntax

```
#define TFTP_IsFileOpenReady UDPIsPutReady(_tftpSocket)
```

Side Effects

None

Returns

true, if it is ok to call TFTP_OpenFile() false, if otherwise.

Description

Macro: bool TFTP_IsFileOpenReady(void)

Checks to see if it is okay to send TFTP file open request to remote server.

Remarks

None

Preconditions

TFTP_Open is already called and TFTP_IsOpened() returned TFTP_OK.

1.9.1.21.1.8 TFTP_IsGetReady Function

File

tftp.h

Syntax

```
TFTP_RESULT TFTP_IsGetReady();
```

Description

This is function TFTP_IsGetReady.

1.9.1.21.1.9 TFTP_IsOpened Function

File

tftp.h

Syntax

```
TFTP_RESULT TFTP_IsOpened();
```

Description

This is function TFTP_IsOpened.

1.9.1.21.1.10 TFTP_IsPutReady Function

File

tftp.h

Syntax

```
TFTP_RESULT TFTP_IsPutReady();
```

Description

This is function TFTP_IsPutReady.

1.9.1.21.1.11 TFTPOpen Function

File

tftp.h

Syntax

```
void TFTPOpen(IP_ADDR * host);
```

Description

This is function TFTPOpen.

1.9.1.21.1.12 TFTPOpenROMFile Macro

File

tftp.h

Syntax

```
#define TFTPOpenROMFile(a,b) TFTPOpenFile((uint8_t *) (a),b)
```

Description

This is macro TFTPOpenROMFile.

1.9.1.21.1.13 TFTP_ACCESS_ERROR Enumeration

File

tftp.h

Syntax

```
typedef enum _TFTP_ACCESS_ERROR {  
    TFTP_ERROR_NOT_DEFINED = 0,  
    TFTP_ERROR_FILE_NOT_FOUND,  
    TFTP_ERROR_ACCESS_VIOLATION,  
    TFTP_ERROR_DISK_FULL,  
    TFTP_ERROR_INVALID_OPERATION,  
    TFTP_ERROR_UNKNOWN_TID,  
    TFTP_ERROR_FILE_EXISTS,  
    TFTP_ERROR_NO_SUCH_USE  
} TFTP_ACCESS_ERROR;
```

Description

Standard error codes as defined by TFTP spec. Use to decode value returned by TFTPGetError().

1.9.1.21.1.14 TFTP_FILE_MODE Enumeration

File

tftp.h

Syntax

```
typedef enum _TFTP_FILE_MODE {  
    TFTP_FILE_MODE_READ = 1,  
    TFTP_FILE_MODE_WRITE = 2  
} TFTP_FILE_MODE;
```

Description

File open mode as used by TFTPFileOpen().

1.9.1.21.1.15 TFTP_RESULT Enumeration

File

tftp.h

Syntax

```
typedef enum _TFTP_RESULT {  
    TFTP_OK = 0,  
    TFTP_NOT_READY,  
    TFTP_END_OF_FILE,  
    TFTP_ERROR,  
    TFTP_RETRY,  
    TFTP_TIMEOUT  
} TFTP_RESULT;
```

Description

Enum. of results returned by most of the TFTP functions.

1.9.1.21.1.16 TFTPGetUploadStatus Function

File

tftp.h

Syntax

```
int8_t TFTPGetUploadStatus();
```

Description

This is function TFTPGetUploadStatus.

1.9.1.21.1.17 TFTP_CHUNK_DESCRIPTOR Type

File

tftp.h

Syntax

```
typedef struct TFTP_CHUNK_DESCRIPTOR@1 TFTP_CHUNK_DESCRIPTOR;
```

Description

This is type TFTP_CHUNK_DESCRIPTOR.

1.9.1.21.1.18 TFTP_UPLOAD_COMPLETE Macro

File

tftp.h

Syntax

```
#define TFTP_UPLOAD_COMPLETE 0
```

Description

Status codes for TFTPGetUploadStatus() function. Zero means upload success, >0 means working and <0 means fatal error.

1.9.1.21.1.19 TFTP_UPLOAD_CONNECT Macro

File

tftp.h

Syntax

```
#define TFTP_UPLOAD_CONNECT 3
```

Description

This is macro TFTP_UPLOAD_CONNECT.

1.9.1.21.1.20 TFTP_UPLOAD_CONNECT_TIMEOUT Macro**File**

tftp.h

Syntax

```
#define TFTP_UPLOAD_CONNECT_TIMEOUT -2
```

Description

This is macro TFTP_UPLOAD_CONNECT_TIMEOUT.

1.9.1.21.1.21 TFTP_UPLOAD_GET_DNS Macro**File**

tftp.h

Syntax

```
#define TFTP_UPLOAD_GET_DNS 1
```

Description

This is macro TFTP_UPLOAD_GET_DNS.

1.9.1.21.1.22 TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT Macro**File**

tftp.h

Syntax

```
#define TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT -1
```

Description

This is macro TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT.

1.9.1.21.1.23 TFTP_UPLOAD_RESOLVE_HOST Macro**File**

tftp.h

Syntax

```
#define TFTP_UPLOAD_RESOLVE_HOST 2
```

Description

This is macro TFTP_UPLOAD_RESOLVE_HOST.

1.9.1.21.1.24 TFTP_UPLOAD_SEND_DATA Macro**File**

tftp.h

Syntax

```
#define TFTP_UPLOAD_SEND_DATA 5
```

Description

This is macro TFTP_UPLOAD_SEND_DATA.

1.9.1.21.1.25 TFTP_UPLOAD_SEND_FILENAME Macro

File

tftp.h

Syntax

```
#define TFTP_UPLOAD_SEND_FILENAME 4
```

Description

This is macro TFTP_UPLOAD_SEND_FILENAME.

1.9.1.21.1.26 TFTP_UPLOAD_SERVER_ERROR Macro

File

tftp.h

Syntax

```
#define TFTP_UPLOAD_SERVER_ERROR -3
```

Description

This is macro TFTP_UPLOAD_SERVER_ERROR.

1.9.1.21.1.27 TFTP_UPLOAD_WAIT_FOR_CLOSURE Macro

File

tftp.h

Syntax

```
#define TFTP_UPLOAD_WAIT_FOR_CLOSURE 6
```

Description

This is macro TFTP_UPLOAD_WAIT_FOR_CLOSURE.

1.9.1.21.2 TFTP Stack Members

Functions and variables intended to be accessed only by the stack

Macros

Name	Description
TFTP_ARP_TIMEOUT_VAL	Number of seconds to wait before declaring TIMEOUT error on Put
TFTP_GET_TIMEOUT_VAL	Number of seconds to wait before declaring TIMEOUT error on Get.
TFTP_MAX_RETRIES	Number of attempts before declaring TIMEOUT error.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.21.2.1 TFTP_ARP_TIMEOUT_VAL Macro

File

tftp.h

Syntax

```
#define TFTP_ARP_TIMEOUT_VAL (3u * TICKS_PER_SECOND)
```

Description

Number of seconds to wait before declaring TIMEOUT error on Put

1.9.1.21.2.2 TFTP_GET_TIMEOUT_VAL Macro

File

tftp.h

Syntax

```
#define TFTP_GET_TIMEOUT_VAL (3u * TICKS_PER_SECOND)
```

Description

Number of seconds to wait before declaring TIMEOUT error on Get.

1.9.1.21.2.3 TFTP_MAX_RETRIES Macro

File

tftp.h

Syntax

```
#define TFTP_MAX_RETRIES (3u)
```

Description

Number of attempts before declaring TIMEOUT error.

1.9.1.21.3 TFTP Internal Members

Functions and variables internal to the TFTP module

Variables

Name	Description
_tftpError	This is variable _tftpError.
_tftpSocket	This is variable _tftpSocket.

Description

The following functions and variables are designated as internal to the TFTP module.

1.9.1.21.3.1 _tftpError Variable

File

tftp.h

Syntax

```
uint16_t _tftpError;
```

Description

This is variable _tftpError.

1.9.1.21.3.2 _tftpSocket Variable

File

tftp.h

Syntax

```
UDP_SOCKET _tftpSocket;
```

Description

This is variable _tftpSocket.

1.9.1.22 Tick Module

Provides accurate time-keeping capabilities.

Description

The Tick module provides accurate time-keeping capabilities based on the hardware clock. By default, it uses Timer 0 on 8-bit parts and Timer 1 on 16- and 32-bit families. The module is interrupt driven, which makes the timing stable and accurate. As such, it is also suitable for a real-time clock.

The Tick module exists to assist with the implementation of non-blocking delays and timeouts. Rather than using a loop to count to a specific number, use the Tick module and compare a previous time with the current time. In this fashion applications can return its unused cycles to the stack during long delays, which increases the overall efficiency of the system.

Tick works best in conjunction with a state machine. In general, call TickGet and store the result. Return to the main stack application, and on future calls compare the current Tick value to the stored one. The constants TICK_SECOND, TICK_MINUTE, and TICK_HOUR can be used to compare against logical time increments.

The following example implements a delay of 0.5 seconds using the Tick module:

```
TICK startTime;

// ...state machine and other states

case SM_SET_DELAY:
    startTime = TickGet();
    sm = SM_DELAY_WAIT;
    return;

case SM_DELAY_WAIT:
    if((LONG)(TickGet() - startTime) < TICK_SECOND/2)
        return;

case SM_DELAY_DONE:
    // This state is entered only after 0.5 second elapses.
```

Ticks are stored internally as 48-bit integers. Using the various TickGet, TickGetDiv256, and TickGetDiv64K functions the Tick is suitable for measuring time increments from a few microseconds to a few years.

If absolute timestamps are required, the SNTP Client module may be more appropriate.

1.9.1.22.1 Tick Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
≡	TickGet	This is function TickGet.
≡	TickGetDiv256	This is function TickGetDiv256.
≡	TickGetDiv64K	This is function TickGetDiv64K.

Macros

Name	Description
TICK_HOUR	Represents one hour in Ticks

TICK_MINUTE	Represents one minute in Ticks
TICK_SECOND	Represents one second in Ticks

Variables

Name	Description
TICK	This is variable TICK.

Description

The following functions and variables are available to the stack application.

1.9.1.22.1.1 TICK Variable**File**

tick.h

Syntax

```
uint32_t TICK;
```

Description

This is variable TICK.

1.9.1.22.1.2 TICK_HOUR Macro**File**

tick.h

Syntax

```
#define TICK_HOUR ((uint64_t)TICKS_PER_SECOND*3600ull)
```

Description

Represents one hour in Ticks

1.9.1.22.1.3 TICK_MINUTE Macro**File**

tick.h

Syntax

```
#define TICK_MINUTE ((uint64_t)TICKS_PER_SECOND*60ull)
```

Description

Represents one minute in Ticks

1.9.1.22.1.4 TICK_SECOND Macro**File**

tick.h

Syntax

```
#define TICK_SECOND ((uint64_t)TICKS_PER_SECOND)
```

Description

Represents one second in Ticks

1.9.1.22.1.5 TickGet Function

File

tick.h

Syntax

```
uint32_t TickGet();
```

Description

This is function TickGet.

1.9.1.22.1.6 TickGetDiv256 Function

File

tick.h

Syntax

```
uint32_t TickGetDiv256();
```

Description

This is function TickGetDiv256.

1.9.1.22.1.7 TickGetDiv64K Function

File

tick.h

Syntax

```
uint32_t TickGetDiv64K();
```

Description

This is function TickGetDiv64K.

1.9.1.22.2 Tick Stack Functions

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
☞	TickInit	XC8
☞	TickUpdate	This is function TickUpdate.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.22.2.1 TickInit Function

File

tick.h

Syntax

```
void TickInit();
```

Description

XC8

1.9.1.22.2.2 TickUpdate Function

File

tick.h

Syntax

```
void TickUpdate();
```

Description

This is function TickUpdate.

1.9.1.22.3 Tick Internal Members

Functions and variables internal to the Tick module

Macros

Name	Description
TICKS_PER_SECOND	Internal core clock drives timer with 1:256 prescaler define TICKS_PER_SECOND (32768ul) // 32kHz crystal drives timer with no scalar

Description

The following functions and variables are designated as internal to the Tick module.

1.9.1.22.3.1 TICKS_PER_SECOND Macro

File

tick.h

Syntax

```
#define TICKS_PER_SECOND ((SYS_CLK_FrequencyPeripheralGet()+128ul)/256ul) // Internal  
core clock drives timer with 1:256 prescaler
```

Description

Internal core clock drives timer with 1:256 prescaler define TICKS_PER_SECOND (32768ul) // 32kHz crystal drives timer with no scalar

1.9.1.23 UDP

Implements the UDP transport layer protocol.

Description

UDP is a standard transport layer protocol described in RFC 768. It provides fast but unreliable data-gram based transfers over networks, and forms the foundation SNTP, SNMP, DNS, and many other protocol standards.

Connections over UDP should be thought of as data-gram based transfers. Each packet is a separate entity, the application should expect some packets to arrive out-of-order or even fail to reach the destination node. This is in contrast to TCP, in which the connection is thought of as a stream and network errors are automatically corrected. These tradeoffs in reliability are made for an increase in throughput. In general, UDP transfers operate 2 to 3 times faster than those made over TCP.

Since UDP is packet-oriented, each packet must be dealt with in its entirety by your application before returning to the main stack loop. When a packet is received, your application will be called to handle it. This packet will no longer be available the next time your application is called, so you must either perform all necessary processing or copy the data elsewhere before returning. When transmitting a packet, your application must build and transmit the complete packet in one cycle.

The UDP flow diagram below provides an overview for the use of the UDP module:

Sockets are opened using UDPOpen. This function can either open a listening socket to wait for incoming segments, or can make a client connection to a remote node. When making a client connection, you will need to perform any required DNS and/or ARP resolution using those modules directly before invoking UDPOpen.

Once the socket is opened, you can immediately begin transmitting data. To transmit a segment, call UDPIsPutReady to determine how many bytes can be written and to designate a currently active socket. Then, use any of the UDPPut family of functions to write data to the socket. Once all data has been written, call UDPFlush to build and transmit the packet. This sequence must be accomplished all in one step. If your application returns to the main stack loop after calling UDPPut but before calling UDPFlush, the data may be lost or the module may behave unpredictably.

To check for received segments, call UDPIsGetReady. If the return value is non-zero, your application must consume the segment by reading data with the UDPGet family. Once all data has been read, return to the main stack loop to wait for an additional segment. UDP segments are only stored for one iteration of the cooperative multi-tasking loop, so your application must complete its processing on a segment or copy it elsewhere before returning. Note that this behavior differs from TCP, which buffers incoming data through multiple stack cycles.

When a socket is no longer needed, call UDPClose to release it back to the pool for future use.

1.9.1.23.1 UDP Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
≡	UDPClose	UDP_SOCKET UDPOpen(UDP_PORT localPort, NODE_INFO *remoteNode, UDP_PORT remotePort);
≡	UDPDiscard	This is function UDPDiscard.
≡	UDPFlush	This is function UDPFlush.
≡	UDPIsGetReady	This is function UDPIsGetReady.
≡	UDPIsPutReady	This is function UDPIsPutReady.
≡	UDPIsOpened	This is function UDPIsOpened.

Macros

Name	Description
INVALID_UDP_PORT	Indicates a UDP port that is not valid
INVALID_UDP_SOCKET	Indicates a UDP socket that is not valid
UDPOpen	Macro of the legacy version of UDPOpen.
UDP_OPEN_IP_ADDRESS	Create a client socket and use dwRemoteHost as a literal IP address.
UDP_OPEN_NODE_INFO	Create a client socket and use dwRemoteHost as a pointer to a NODE_INFO structure containing the exact remote IP address and MAC address to use.
UDP_OPEN_RAM_HOST	Emit an undeclared identifier diagnostic if code tries to use UDP_OPEN_RAM_HOST while the DNS client module is not enabled.
UDP_OPEN_ROM_HOST	Emit an undeclared identifier diagnostic if code tries to use UDP_OPEN_ROM_HOST while the DNS client module is not enabled.
UDP_OPEN_SERVER	Create a server socket and ignore dwRemoteHost.

Types

Name	Description
UDP_SOCKET	Provides a handle to a UDP Socket

Description

The following functions and variables are available to the stack application.

1.9.1.23.1.1 INVALID_UDP_PORT Macro

File

udp.h

Syntax

```
#define INVALID_UDP_PORT (0ul) // Indicates a UDP port that is not valid
```

Description

Indicates a UDP port that is not valid

1.9.1.23.1.2 INVALID_UDP_SOCKET Macro

File

udp.h

Syntax

```
#define INVALID_UDP_SOCKET (0xffu) // Indicates a UDP socket that is not valid
```

Description

Indicates a UDP socket that is not valid

1.9.1.23.1.3 UDP_SOCKET Type

File

udp.h

Syntax

```
typedef uint8_t UDP_SOCKET;
```

Description

Provides a handle to a UDP Socket

1.9.1.23.1.4 UDPOpen Macro

Macro of the legacy version of UDPOpen.

File

udp.h

Syntax

```
#define UDPOpen(localPort,remoteNode,remotePort)  
UDPOpenEx((uint32_t)remoteNode,UDP_OPEN_NODE_INFO,localPort,remotePort)
```

Description

UDPOpen is a macro replacement of the legacy implementation of UDPOpen. Creates a UDP socket handle for transmitting or receiving UDP packets. Call this function to obtain a handle required by other UDP function.

Remarks

When finished using the UDP socket handle, call the UDPClose() function to free the socket and delete the handle.

Preconditions

UDPInit() must have been previously called.

Parameters

Parameters	Description
localPort	UDP port number to listen on. If 0, stack will dynamically assign a unique port number to use.
remoteNode	Pointer to remote node info (MAC and IP address) for this connection. If this is a server socket (receives the first packet) or the destination is the broadcast address, then this parameter should be NULL.
remotePort	For client sockets, the remote port number.

Return Values

Return Values	Description
Success	A UDP socket handle that can be used for subsequent UDP API calls.
Failure	INVALID_UDP_SOCKET. This function fails when no more UDP socket handles are available. Increase MAX_UDP_SOCKETS to make more sockets available.

Function

```
UDP_SOCKET UDPOpen(UDP_PORT localPort, NODE_INFO* remoteNode,  
                  UDP_PORT remotePort)
```

1.9.1.23.1.5 UDPClose Function**File**

udp.h

Syntax

```
void UDPClose(UDP_SOCKET s);
```

Description

UDP_SOCKET UDPOpen(UDP_PORT localPort, NODE_INFO *remoteNode, UDP_PORT remotePort);

1.9.1.23.1.6 UDPDiscard Function**File**

udp.h

Syntax

```
void UDPDiscard();
```

Description

This is function UDPDiscard.

1.9.1.23.1.7 UDPFlush Function**File**

udp.h

Syntax

```
void UDPFlush();
```

Description

This is function UDPFlush.

1.9.1.23.1.8 UDPIsGetReady Function

File

udp.h

Syntax

```
uint16_t UDPIsGetReady(UDP_SOCKET s);
```

Description

This is function UDPIsGetReady.

1.9.1.23.1.9 UDPIsPutReady Function

File

udp.h

Syntax

```
uint16_t UDPIsPutReady(UDP_SOCKET s);
```

Description

This is function UDPIsPutReady.

1.9.1.23.1.10 UDPIsOpened Function

File

udp.h

Syntax

```
bool UDPIsOpened(UDP_SOCKET socket);
```

Description

This is function UDPIsOpened.

1.9.1.23.1.11 UDP_OPEN_IP_ADDRESS Macro

File

udp.h

Syntax

```
#define UDP_OPEN_IP_ADDRESS 3u
```

Description

Create a client socket and use dwRemoteHost as a literal IP address.

1.9.1.23.1.12 UDP_OPEN_NODE_INFO Macro

File

udp.h

Syntax

```
#define UDP_OPEN_NODE_INFO 4u
```

Description

Create a client socket and use dwRemoteHost as a pointer to a NODE_INFO structure containing the exact remote IP address and MAC address to use.

1.9.1.23.1.13 UDP_OPEN_RAM_HOST Macro

File

udp.h

Syntax

#define UDP_OPEN_RAM_HOST You_need_to_enable_STACK_USE_DNS_to_use_UDP_OPEN_RAM_HOST

Description

Emit an undeclared identifier diagnostic if code tries to use UDP_OPEN_RAM_HOST while the DNS client module is not enabled.

1.9.1.23.1.14 UDP_OPEN_ROM_HOST Macro

File

udp.h

Syntax

#define UDP_OPEN_ROM_HOST You_need_to_enable_STACK_USE_DNS_to_use_UDP_OPEN_ROM_HOST

Description

Emit an undeclared identifier diagnostic if code tries to use UDP_OPEN_ROM_HOST while the DNS client module is not enabled.

1.9.1.23.1.15 UDP_OPEN_SERVER Macro

File

udp.h

Syntax

#define UDP_OPEN_SERVER 0u

Description

Create a server socket and ignore dwRemoteHost.

1.9.1.23.2 UDP Stack Members

Functions and variables intended to be accessed only by the stack

Functions

	Name	Description
≡	UDPInit	
≡	UDPTask	This is function UDPTask.

Description

The following functions and variables are public, but are intended only to be accessed by the stack itself. Applications should generally not call these functions or modify these variables.

1.9.1.23.2.1 UDPInit Function

File

udp.h

Syntax

void UDPInit();

Section

Function Prototypes

1.9.1.23.2.2 UDPTask Function

File

udp.h

Syntax

```
void UDPTask( ) ;
```

Description

This is function UDPTask.

1.9.1.23.3 UDP Internal Members

Functions and variables internal to the UDP module

Types

Name	Description
UDP_HEADER	Stores the header of a UDP packet
UDP_PORT	Stores a UDP Port Number
UDP_SOCKET_INFO	Stores information about a current UDP socket

Variables

Name	Description
activeUDPSocket	
UDPRxCount	This is variable UDPRxCount.
UDPSocketInfo	This is variable UDPSocketInfo.
UDPTxCount	This is variable UDPTxCount.

Description

The following functions and variables are designated as internal to the UDP module.

1.9.1.23.3.1 activeUDPSocket Variable

File

udp.h

Syntax

```
UDP_SOCKET activeUDPSocket ;
```

Section

External Global Variables

1.9.1.23.3.2 UDP_HEADER Type

File

udp.h

Syntax

```
typedef struct UDP_HEADER@1 UDP_HEADER ;
```

Description

Stores the header of a UDP packet

1.9.1.23.3.3 UDP_PORT Type

File

udp.h

Syntax

```
typedef uint16_t UDP_PORT;
```

Description

Stores a UDP Port Number

1.9.1.23.3.4 UDP_SOCKET_INFO Type

File

udp.h

Syntax

```
typedef struct UDP_SOCKET_INFO@1 UDP_SOCKET_INFO;
```

Description

Stores information about a current UDP socket

1.9.1.23.3.5 UDPRxCount Variable

File

udp.h

Syntax

```
uint16_t UDPRxCount;
```

Description

This is variable UDPRxCount.

1.9.1.23.3.6 UDPSocketInfo Variable

File

udp.h

Syntax

```
UDP_SOCKET_INFO UDPSocketInfo[MAX_UDP_SOCKETS];
```

Description

This is variable UDPSocketInfo.

1.9.1.23.3.7 UDPTxCount Variable

File

udp.h

Syntax

```
uint16_t UDPTxCount;
```

Description

This is variable UDPTxCount.

1.9.2 Wi-Fi API

Describes the API for controlling the MRF24WB0M / MRF24WG0M Wi-Fi API

Description

Feature similarities and differences of MRF24WB0M and MRF24WG0M

MRF24WB0M	MRF24WG0M
1 , 2 Mbps	802.11b and 802.11g rates
Ad-hoc (OPEN, WEP)	Ad-hoc (OPEN, WEP)
Infrastructure (OPEN, WEP, WPA/WPA2-PSK)	Infrastructure (OPEN, WEP, WPA-PSK/WPA2-PSK, WPS)
	SoftAP (OPEN, WEP40/104)
	Wi-Fi Direct (P2P)

Unlike Ethernet, a Wi-Fi 802.11 application needs to initiate a connection to an access point, Wi-Fi Direct or an ad hoc network, before data communications can commence.

The `drv_wifi_config.h` file has several compile-time constants that can be customized (e.g. `MY_DEFAULT_SSID_NAME`) as needed.

In order to initiate a connection there is a sequence of steps that should be followed.

1) A connection profile must be created (see `WF_CPCreate()`). The connection profile contains information directing the WiFi driver about the nature of the connection that will be established. The connection profile defines:

- a. SSID (name of Access Point)
- b. Security (open, WEP, WPA, WPA2, WPS-PBC, WPS-PIN)
- c. Network type (Infrastructure, Ad-hoc, SoftAP, Wi-Fi Direct).

The Connection Profile functions are used to create and define an connection profile. These functions all begin with `WF_CP...`

2) The connection algorithm must be defined, and applies to all connection profiles. For most applications the defaults will be sufficient. For example, the default connection algorithm channel list for scanning is 1,2,3,4,5,6,7,8,9,10, and 11. However, if, in your application you know the Access Point will always be on channel 6 you could change this setting, thus making the scan process more efficient. Functions pertaining to the connection algorithm all begin with `WF_CA...`

3) Once a connection profile and the connection algorithm are customized for an application, the `WF_CMConnect()` function must be called to initiate the connection process.

4) After `WF_Connect()` is called the host application will be notified when the MRF24WB0M / MRF24WG0M has succeeded (or failed) in establishing a connection via the event mechanism. The `drv_wifi_config.c` file has a function, `WF_ProcessEvent()`, that is a template for processing MRF24WB0M / MRF24WG0M events. In the WiFi demos it simply prints to the console (if the UART is enabled) that the event occurred. This file can be modified to suit the needs of an application – for example, an application could pend on a global flag that would be set in `WF_ProcessEvent()` when the connection succeeded. Please refer to `WF_ProcessEvent` for more information on WiFi event handling.

WF_ProcessEvent()

This function is called by the Wi-Fi Driver when an event occurs that the host CPU needs to be notified of. There are several Wi-Fi connection related events that the application can choose whether to be notified or not. And, there are several events

the application will always be notified of.

The function `WF_ProcessEvent()` can be customized to support desired handling of events.

The MRF2WB0M / MRF24WG0M demos (under the TCPIP Demo App, TCPIP WiFi Console, TCPIP WiFi EasyConfig and TCPIP WiFi G directories) contain a function, `WF_Connect()`, in `main.c` that executes the above steps and can be referred to as an example of how to initiate a WiFi connection.

Below describes the host API to the MRF24WB0M / MRF24WG0M on-chip connection manager which creates and maintains Wi-Fi connections. The API is divided into these major sections:

API Section	Description
Wi-Fi Compilation Options	Describes the various Wi-Fi compilation options
Initialization	Functions to initialize the host API and MRF24WB0M / MRF24WG0M
Wi-Fi Network Topologies	Describes the various Wi-Fi network topologies
Wi-Fi Connection Profile	Functions to create and maintain one or more connection profiles
Wi-Fi Connection Algorithm	Functions to fine tune the connection algorithm
Wi-Fi Connection Manager	Functions to start and stop an 802.11 connection
Wi-Fi Scan	Functions to scan for wireless networks
Wi-Fi Security	Functions to handle wireless 802.11 security
Wi-Fi Tx Power Control	Functions to control the MRF24WB0M / MRF24WG0M Tx power
Wi-Fi Power Save	Functions to save power consumption by the MRF24WB0M / MRF24WG0M
Wi-Fi Driver Management Functions	Functions to provide access to the MRF24W Wi-Fi controller
Wi-Fi Miscellaneous	Functions to set a custom MAC address, get device information, etc.
WF_ProcessEvent	Functions to handle events from the MRF24WB0M / MRF24WG0M

SPI

The `drv_wifi_spi.c` file contains functions that the Wi-Fi Driver will use to initialize, send, and receive SPI messages between the host CPU and the MRF24WB0M / MRF24WG0M. To communicate with the MRF24WB0M / MRF24WG0M, which is always an SPI slave, the host CPU SPI controller needs to be configured as follows:

- Mode = 0
- CPOL (clock polarity) = 0
- CPHA (clock phase) = 0
- Host CPU set as master
- Clock idles high
- 8-bit transfer length
- Data changes on falling edge
- Data sampled on rising edge

Below is a list of functions in `drv_wifi_spi.c` that must be customized for the specific host CPU architecture:

Function	Description
<code>WF_SpiInit()</code>	Initializes the host CPU SPI controller for usage by the Wi-Fi driver. Called by the Wi-Fi driver during initialization.
<code>WF_SpiTxRx()</code>	Transmits and/or receives SPI data from the MRF24WB0M / MRF24WG0M.

<code>WF_SpiEnableChipSelect()</code>	Set slave select line on MRF24WB0M / MRF24WG0M low (start SPI transfer). If SPI bus is shared with any other devices then this function also needs to save the current SPI context and then configure the MRF24WB0M / MRF24WG0M SPI context.
<code>WF_SpiDisableChipSelect()</code>	Set slave select line on MRF24WB0M / MRF24WG0M high (end SPI transfer). If SPI bus is shared with any other devices then this function also needs to restore the SPI context (saved during <code>WF_SpiEnableChipSelect()</code>).

External Interrupt

The `drv_wifi_eint.c` file contains functions that the Wi-Fi Driver will use to enable and disable the MRF24WB0M / MRF24WG0M external interrupt as well as get interrupt status. The functions in this module need to be customized for the specific host CPU architecture.

The MRF24WB0M / MRF24WG0M asserts its EXINT (external interrupt) line (active low) when specific events occur, such as a data message being received. Note that the host CPU has a choice to either configure the EXINT line to generate an actual interrupt, or, it can be polled. Below is a list of the Wi-Fi Driver functions within `drv_wifi_eint.c` that must be customized for the specific Host CPU architecture.

Function	Description
<code>WF_EintInit()</code>	Configures the interrupt for use and leaves it in a disabled state. Will be called by the Wi-Fi driver during initialization. If polling the EXINT pin then this function won't have any work to do except leave the interrupt in a logically disabled state.
<code>WF_EintEnable()</code>	Enables the MRF24WB0M / MRF24WG0M external interrupt. If using real interrupts then enable the interrupt. If polling the EXINT pin then this function enables polling of the pin.
<code>WF_EintDisable()</code>	Disables the MRF24WB0M / MRF24WG0M external interrupt. If using real interrupts then disable the interrupt. If polling the EXINT pin then this function disables polling of the pin.
<code>WF_EintIsr()</code>	This is the interrupt service routine invoked when the EXINT line goes low. It should perform any necessary housekeeping, such as clearing the interrupt. The interrupt must remain disabled until the Wi-Fi Driver calls <code>WF_EintEnable()</code> . The Wi-Fi driver function, <code>WFEintHandler()</code> must be called.
<code>WF_EintIsDisabled()</code>	Returns true if the external interrupt is disabled, else returns false.
<code>WFEintHandler()</code>	This function does not need to be customized – it is part of the Wi-Fi driver. However, it is added to this list because it must be called each time the MRF24WB0M / MRF24WG0M interrupt service routine (ISR) occurs.

1.9.2.1 Wi-Fi Network Topologies

Describes the various Wi-Fi network topologies

Description

This section describes the various Wi-Fi network topologies

- Infrastructure
- Ad-hoc
- SoftAP
- Wi-Fi Direct

1.9.2.1.1 Infrastructure Network

Describes the infrastructure network and how to set up MRF24W in infrastructure network mode

Description

Configure MRF24W as client in an Infrastructure (CFG_WF_INFRASTRUCTURE) network

In `drv_wifi_config.h`, to start up as a client in an infrastructure network, define

- MY_DEFAULT_NETWORK_TYPE as CFG_WF_INFRASTRUCTURE.
- MY_DEFAULT_SSID_NAME

This SSID will be the AP/router's SSID that you wish to be connected to.

- MY_DEFAULT_WIFI_SECURITY_MODE

This security mode is the AP/router's wireless security mode.

Infrastructure Network Topology

Below shows an example of the infrastructure network.

This infrastructure network shows a laptop computer and the MRF24W communicating with each other through a wireless access point (AP) and router. This network can gain access to the internet if the AP/router is connected to a WAN.

1.9.2.1.2 Ad-hoc Network

Describes the ad-hoc network and how to set up MRF24W in ad-hoc network mode

Description

Configure MRF24W as Ad-hoc (CFG_WF_ADHOC) device

In `drv_wifi_config.h`, to start up as a ad-hoc device, define

- MY_DEFAULT_NETWORK_TYPE as CFG_WF_ADHOC.
- MY_DEFAULT_SSID_NAME

Example "MCHPAdhoc_123".

Ad-hoc Network Topology

Below shows an example of the Ad-hoc network.

In this example, we assume that the Microchip development board with MRF24WB0MA/B or MRF24WG0MA/B is the first station to broadcast that it wants to create the network (and it is successfully able to do so). In this case, the laptop will then join the ad-hoc network after the MRF24W has gone through the steps of setting up the ad-hoc network.

The security mode supported is open mode and WEP security.

According to specifications, ad-hoc only supports 802.11b rates of 1, 2, 5.5 and 11 Mbps.

Android devices do not support ad-hoc network.

1.9.2.1.3 SoftAP Network

Describes the softAP network and how to set up MRF24WG0M in softAP network mode

Description

This is only supported by MRF24WG0M.

SoftAP mode is only supported by MLA v5.42 July 2012 releases or later.

SoftAP tracking of clients' status (dhcps.c) is only supported by MLA v5.42.06 Mar 2013 releases or later.

Configure MRF24W as softAP (CFG_WF_SOFT_AP)

In `drv_wifi_config.h`, to start up as a softAP, define

- MY_DEFAULT_NETWORK_TYPE as CFG_WF_SOFT_AP.
- MY_DEFAULT_SSID_NAME

Example "MCHPSoftAP_123".

- MY_DEFAULT_CHANNEL LIST

This specifies the channel MRF24W softAP will reside in.

SoftAP Network Topology

Below shows an example of the softAP network.

SoftAP (software enabled AP) functions can be used to extend wireless coverage and share internet connection with others.

Clients supported in SoftAP Mode

For MRF24WG0M FW version 0x3107, softAP can only support 1 client.

For future MRF24WG0M FW versions, softAP can support up to a max of 4 clients.

SoftAP Operations : Support of Max 1 Client Scenario

Once the first client is connected to MRF24WG0M SoftAP, softAP will remember client's MAC address. Only when the client does a disconnect in the 2 scenarios below, softAP will reset the MAC address to NULL. When this happens, another client can connect to MRF24WG0M softAP.

Below are 2 scenarios in which a client disconnects from MRF24WG0M SoftAP

Scenario A

Client A does a proper disconnection, that is disassociation & disauthentication frames are sent. MRF24WG0M SoftAP, after receiving these frames, will reset the MAC address to NULL. Another client B can then connect to our SoftAP.

Scenario B

Client just powers off, in other words, did NOT inform MRF24WG0M SoftAP it is disconnecting.

(SOFTAP_CHECK_LINK_STATUS) To cater to this situation, MRF24WG0M SoftAP will ping STA by transmitting NULL DATA frames to STA to check whether STA is alive/dead. If STA is alive, it will respond to the NULL DATA frames received by transmitting ACK frames back to softAP. If STA is dead, softAP will not receive any frames from this particular device. Once the PARAM_LINK_DOWN_THRESHOLD is reached, softAP considers the device to be dead. Refer to function prototype `WF_SetLinkDownThreshold()`.

SoftAP Operations : Support of Max 4 Clients Scenario

To know how many clients and their connection status, `dhcps.c` has a data struct `DHCP_IP_POOL` and the variable `DhcpIpPool[]` that keeps track of clients connected to softAP, such as client's MAC address, IP address.

```
typedef struct
{
    MAC_ADDR ClientMAC;
    IP_ADDR Client_Addr;
    BOOL isUsed;
    UINT32 Client_Lease_Time;
}DHCP_IP_POOL;

DHCP_IP_POOL DhcpIpPool[MAX_DHCP_CLIENTS_NUMBER];
```

However, be aware there would be a latency in client's status. For example, a client has disconnected from softAP. But it would take some time before this update in status is reflected in DhcpIpPool[].

Detection of SoftAP's SSID

Certain devices may only support active scan. Based on 802.11 specifications, passive scan is mandatory but active scan is optional.

MRF24WG0M FW version 0x3107 softAP only supports passive scan. Such devices may not be able to detect MRF24WG0M softAP.

MRF24WG0M FW version 0x3108 and later softAP supports both passive and active scan. If your device is unable to detect MRF24WG0M softAP SSID, check your MRF24WG0M FW version.

Consideration of SoftAP consuming more transmit power

According to 802.11 specifications, APs are expected to transmit beacons during beacon intervals (BI), thereby consuming more transmit power, as compared to being a client in infrastructure network type.

MRF24W SoftAP channel setting

MY_DEFAULT_CHANNEL_LIST will indicate the channel the MRF24W softAP will reside in.

For example,

```
#define MY_DEFAULT_CHANNEL_LIST {6}
```

means that MRF24W softAP will reside in social channel 6.

It is recommended that social channel 1 or 6 or 11 be used for MRF24W softAP channel setting.

- **Why does the software hangs at WF_ProcessEvent() in line WF_ASSERT(FALSE) when in softAP network type ?**

A possible cause could be the handling of WF event WF_EVENT_SOFT_AP_EVENT. This new feature is only available for MRF24WG0M (i) FW version 0x3108 and later and (ii) MLA v5.42.06 release or later. If you are using MRF24WG0M FW version 0x3108 or later and at the same time using prior to MLA v5.42.06 release, MRF24WG0M is generating this event, however WF_ProcessEvent() did not handle this event and fall into source code line (default: WF_ASSERT(FALSE)). The corrective action is to port over this event handling in drv_wifi_event_handler.c and WF_ProcessEvent() (TCPIPO WiFi EasyConfig Demo) from MLA v5.42.06 release or later.

- **Why does the MRF24W in softAP network type, with IP address as 192.168.1.1, on certain wireless network has problems with DHCP client assigning new IP address upon network redirection ?**

192.168.1.1 is a common IP address with most APs. Potential conflict can arise when there are 2 active DHCP servers in the same wireless network (i.e. AP DHCP server and MRF24W DHCP server). When network redirection is executed, the TCPIP SW may still have the MRF24W DHCP server still active, creating conflicting presence of 2 DHCP servers. This may require stack software change in TCPIP stack to be able to disable the local MRF24W DHCP server after network redirection.

1.9.2.1.4 Wi-Fi Direct Network

Describes the Wi-Fi Direct network and how to set up MRF24WG0M in Wi-Fi Direct network mode

Description

This is only supported by MRF24WG0M as a group client (GC) in a Wi-Fi Direct network type.

Configure MRF24W as Wi-Fi DIRECT (CFG_WF_P2P) Group Client (GC)

In `drv_wifi_config.h`, to start up as a Wi-Fi direct group client device, define

- `MY_DEFAULT_NETWORK_TYPE` as `CFG_WF_P2P`.
- `MY_DEFAULT_SSID_NAME` as "DIRECT-"

which is an unique specifier to identify a Wi-Fi Direct network.

- `MY_DEFAULT_CHANNEL_LIST` as {1, 6, 11}

which are specified social channels for Wi-Fi Direct network.

Wi-Fi DIRECT Network Topology

Below shows an example of the WiFi Direct (peer-to-peer P2P) network.

WiFi Direct does not support 802.11b rates and therefore, only MRF24WG0MA/B is able to support such network type.

Wi-Fi Direct allows you to configure a secured wireless network between several devices, such as smart devices, laptops / computers with wireless network adaptors, without using an access point. Wi-Fi Direct supports WPS (WiFi Protected Setup) connection method, which is known as the WSC (WiFi Simple Configuration) Config Methods in the Wi-Fi Peer-to-Peer (P2P) Technical Specifications, in particular WPS push button method with WPA2-PSK.

From negotiation process, each device will determine which devices become GO (group owner) or GC (group client). The `GroupOwnerIntent` field in the P2P IE (information element) will indicate the level of desire to become the GO. The higher the value, the higher the desire to be the GO. Since MRF24WG0MA/B supports the role of GC only, it implies `GroupOwnerIntent=0` (P2P IE).

Within each Wi-Fi Direct network, there can be only 1 group owner, similar to only single AP in the infrastructure network.

1.9.2.2 Wi-Fi Connection Profile

Functions to setup, use, and teardown connection profiles

Description

This section describes the API functions related to creating and using connection profiles. At least one connection profile must be created. The connection profile defines elements required by the MRF24WB0M / MRF24WG0M to establish a connection to a Wi-Fi network.

Modifying Connection Profile after Connection is Established


A connection profile can be updated while it is being used for an active connection. However, the updates do not take effect until one of the following occurs:

- Connection is disabled and re-enabled by the host application
- Connection algorithm loses the connection, exhausts all retries, and then reloads the connection profile.

To ensure that connection profile updates take place at a known point in time it is recommended that the host application call `WF_CMDDisconnect()`, update the connection profile, then call `WF_CMConnect()`.

1.9.2.2.1 Connection Profile Public Members

Structures

	Name	Description
	WFCPElementsStruct	Connection profile elements structure

Description

1.9.2.2.1.1 WFCPElementsStruct Structure

File

drv_wifi_api.h

Syntax

```
struct WFCPElementsStruct {
    uint8_t ssid[WF_MAX_SSID_LENGTH];
    uint8_t bssid[WF_BSSID_LENGTH];
    uint8_t ssidLength;
    uint8_t securityType;
    uint8_t securityKey[WF_MAX_SECURITY_KEY_LENGTH];
    uint8_t securityKeyLength;
    uint8_t wepDefaultKeyId;
    uint8_t networkType;
    uint8_t adHocBehavior;
    uint8_t hiddenSSID;
    uint8_t wepKeyType;
};
```

Members

Members	Description
uint8_t ssid[WF_MAX_SSID_LENGTH];	SSID, which must be less than or equal to 32 characters. Set to all 0's if not being used. If ssidLength is 0 this field is ignored. If SSID is not defined then the MRF24W, when using this profile to connect, will scan all channels within its regional domain. Default: SSID not used.
uint8_t bssid[WF_BSSID_LENGTH];	Basic Service Set Identifier, always 6 bytes. This is the 48-bit MAC of the SSID. It is an optional field that can be used to specify a specific SSID if more than one AP exists with the same SSID. This field can also be used in lieu of the SSID. Set each byte to 0xFF if BSSID is not going to be used. Default: BSSID not used (all FF's)
uint8_t ssidLength;	Number of ASCII bytes in ssid. Set to 0 is SSID is not going to be used. Default: 0

uint8_t securityType;	Designates the desired security level for the connection. Choices are:
uint8_t securityKey[WF_MAX_SECURITY_KEY_LENGTH];	Set to NULL if securityType is WF_SECURITY_OPEN. If securityKeyLength is 0 this field is ignored.
uint8_t securityKeyLength;	Number of bytes used in the securityKey. Set to 0 if securityType is WF_SECURITY_OPEN.
uint8_t wepDefaultKeyId;	This field is only used if securityType is WF_SECURITY_WEP_40 or WF_SECURITY_WEP_104. This field designates which of the four WEP keys defined in securityKey to use when connecting to a WiFi network. Only WEP key index (wepDefaultKeyId) 0 is used in RF module FW.
uint8_t networkType;	WF_INFRASTRUCTURE / WF_ADHOC / WF_P2P / WF_SOFT_AP Default: WF_INFRASTRUCTURE
uint8_t adHocBehavior;	Only applicable if networkType is WF_ADHOC. Configures Adhoc behavior. Choices are:
uint8_t hiddenSSID;	1 - enable hidden ssid in adhoc mode
uint8_t wepKeyType;	0- shared key, 1 - open key

Description

Connection profile elements structure

1.9.2.2.2 Connection Profile Internal Members

Description

1.9.2.3 Wi-Fi Connection Algorithm

Functions to alter the behavior of the connection process



Description

The connection algorithm is used to fine-tune the MRF24WB0M / MRF24WG0M algorithm used in the connection process. The connection algorithm can only be changed when the MRF24WB0M / MRF24WG0M is not connected to an 802.11 network.


1.9.2.3.1 Connection Algorithm Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
	WF_CAGetElements	Reads all Connection Algorithm elements.
	WF_CASetElements	Writes all Connection Algorithm elements.

Structures

	Name	Description
	WFCAElementsStruct	Connection Algorithm Elements

Description

The following functions and variables are available to the stack application.

1.9.2.3.1.1 WF_CAGetElements Function

Reads all Connection Algorithm elements.

File

drv_wifi_api.h

Syntax

```
void WF_CAGetElements(tWFCAElements * p_elements);
```

Returns

None

Description

Sends a message to the MRF24W which requests all the Connection Algorithm elements.

Remarks

None

Preconditions

MACInit must be called first.

Parameters

Parameters	Description
tWFCAElements * p_elements	Pointer to the output structure (tWFCAElements) where the connection algorithm elements are written.

Function

```
void WF_CAGetElements(tWFCAElements *p_elements)
```

1.9.2.3.1.2 WF_CASetElements Function

Writes all Connection Algorithm elements.

File

drv_wifi_api.h

Syntax

```
void WF_CASetElements(tWFCAElements * p_elements);
```

Returns

None

Description

Sends a message to the MRF24W which sets all the Connection Algorithm elements.

Remarks

None

Preconditions

MACInit must be called first.

Parameters

Parameters	Description
tWFCAElements * p_elements	Pointer to the input structure (tWFCAElements) containing the connection algorithm elements.

Function

```
void WF_CASetElements(tWFCAElements *p_elements)
```

1.9.2.3.1.3 WFCACElementsStruct Structure**File**

```
drv_wifi_api.h
```

Syntax

```
struct WFCACElementsStruct {
    uint16_t listenInterval;
    uint8_t scanType;
    uint8_t rssi;
    uint8_t connectionProfileList[WF_CP_LIST_LENGTH];
    uint8_t listRetryCount;
    uint8_t eventNotificationAction;
    uint8_t beaconTimeoutAction;
    uint8_t deauthAction;
    uint8_t channelList[WF_CHANNEL_LIST_LENGTH];
    uint8_t numChannelsInList;
    uint8_t beaconTimeout;
    uint8_t scanCount;
    uint8_t pad1;
    uint16_t minChannelTime;
    uint16_t maxChannelTime;
    uint16_t probeDelay;
    uint16_t dtimInterval;
    uint16_t beaconPrd;
};
```

Members

Members	Description
uint16_t listenInterval;	<p>This parameter is only used when PS Poll mode is enabled. See WF_PsPollEnable(). Number of 100ms intervals between instances when the MRF24W wakes up to received buffered messages from the network. Range is from 1 (100ms) to 6553.5 sec (~109 min).</p> <p>Note that the 802.11 standard defines the listen interval in terms of Beacon Periods, which are typically 100ms. If the MRF24W is communicating to a network with a network that has Beacon Periods that is not 100ms it will round up (or down) as needed to match the actual Beacon Period as closely as possible.</p> <p>Important Note: If the listenInterval is modified while connected to a network the MRF24W will automatically reconnect to the network with the new Beacon Period value. This may cause a temporary loss of data packets.</p>
uint8_t scanType;	<p>WF_ACTIVE_SCAN (Probe Requests transmitted out) or WF_PASSIVE_SCAN (listen only for beacons received)</p> <p>Default: WF_ACTIVE_SCAN</p>
uint8_t rssi;	<p>Specifies RSSI restrictions when connecting. This field is only used if:</p> <ol style="list-style-type: none"> 1. The Connection Profile has not defined a SSID or BSSID, or 2. An SSID is defined in the Connection Profile and multiple AP's are discovered with the same SSID.
uint8_t connectionProfileList[WF_CP_LIST_LENGTH];	<p>Note: Connection Profile lists are not yet supported. This array should be set to all FF's.</p>

uint8_t listRetryCount;	This field is used to specify the number of retries for the single connection profile before taking the connection lost action. Range 1 to 254 or WF_RETRY_FOREVER (255) Default is 3
uint8_t eventNotificationAction;	There are several connection-related events that can occur. The Host has the option to be notified (or not) when some of these events occur. This field controls event notification for connection-related events.
uint8_t beaconTimeoutAction;	Specifies the action the Connection Manager should take if a Connection is lost due to a Beacon Timeout. If this field is set to WF_ATTEMPT_TO_RECONNECT then the number of attempts is limited to the value in listRetryCount. Choices are: WF_ATTEMPT_TO_RECONNECT or WF_DO_NOT_ATTEMPT_TO_RECONNECT Default: WF_ATTEMPT_TO_RECONNECT
uint8_t deauthAction;	Designates what action the Connection Manager should take if it receives a Deauthentication message from the AP. If this field is set to WF_ATTEMPT_TO_RECONNECT then the number of attempts is limited to the value in listRetryCount. Choices are: WF_ATTEMPT_TO_RECONNECT or WF_DO_NOT_ATTEMPT_TO_RECONNECT Default: WF_ATTEMPT_TO_RECONNECT
uint8_t channelList[WF_CHANNEL_LIST_LENGTH];	List of one or more channels that the MRF24W should utilize when connecting or scanning. If numChannelsInList is set to 0 then this parameter should be set to NULL. Default: All valid channels for the regional domain of the MRF24W (set at manufacturing).
uint8_t numChannelsInList;	Number of channels in channelList. If set to 0 then the MRF24W will populate the list with all valid channels for the regional domain. Default: The number of valid channels for the regional domain of the MRF24W (set at manufacturing).
uint8_t beaconTimeout;	Specifies the number of beacons that can be missed before the action described in beaconTimeoutAction is taken.
uint8_t scanCount;	The number of times to scan a channel while attempting to find a particular access point. Default: 1
uint16_t minChannelTime;	The minimum time (in milliseconds) the connection manager will wait for a probe response after sending a probe request. If no probe responses are received in minChannelTime then the connection manager will go on to the next channel, if any are left to scan, or quit. Default: 200ms
uint16_t maxChannelTime;	If a probe response is received within minChannelTime then the connection manager will continue to collect any additional probe responses up to maxChannelTime before going to the next channel in the channelList. Units are in milliseconds. Default: 400ms
uint16_t probeDelay;	The number of microseconds to delay before transmitting a probe request following the channel change event. Default: 20us
uint16_t dtimInterval;	Default : 4
uint16_t beaconPrd;	Default : 100 (ms)

Description

Connection Algorithm Elements

1.9.2.3.2 Connection Algorithm Internal Members

Functions and variables internal to the module

Description

The following functions and variables are designated as internal to the module.

1.9.2.4 Wi-Fi Connection Manager

Functions to manage the connection process

Description

The connection manager uses the connection algorithm and one or more connection profiles to connect to a network.

2 options are offered

1. Connection Manager Handled in Host Stack Software

MRF24W FW has a built-in connection manager, and it is enabled by default. If the host stack software is developed to have its own independent connection manager, the MRF24W connection manager should be disabled to avoid some possible conflicts.

In drv_wifi_config.h, enable definition #define DISABLE_MODULE_FW_CONNECT_MANAGER_IN_INFRASTRUCTURE

2 Wi-Fi APIs that are affected if MRF24W connection manager is not disabled

A) UINT16 WF_CMDDisconnect(void)

B) UINT16 WF_Scan(UINT8 CpId)

For MRF24WB0M with FW versions older than 0x120C, the potential conflict between the 2 connection managers in host stack software and MRF24W firmware can cause fatal FW crash in MRF24WB FW.

2. Connection Manager Handled Entirely by MRF24W FW

Utilizes MRF24W FW built-in connection manager. This is enabled by default.

In drv_wifi_config.h, make sure to disable definition #define
DISABLE_MODULE_FW_CONNECT_MANAGER_IN_INFRASTRUCTURE

1.9.2.4.1 Connection Manager Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
≡	WF_CMDDisconnect	Commands the MRF24W to close any open connections and/or to cease attempting to connect.
≡	WF_CMInfoGetFSMStats	This is function WF_CMInfoGetFSMStats.
≡	WF_CMGetConnectContext	Retrieves WF connection context for MRF24WG0MA/B
≡	WF_DisableModuleConnectionManager	When compilation flag WF_MODULE_CONNECTION_MANAGER is set to WF_DISABLED, this will disable MRF24W connection manager.

Description

The following functions and variables are available to the stack application.

1.9.2.4.1.1 WF_CMDDisconnect Function

Commands the MRF24W to close any open connections and/or to cease attempting to connect.

File

drv_wifi_api.h

Syntax

```
uint16_t WF_CMDDisconnect();
```

Returns

Operation results. Success or Failure

Description

Directs the Connection Manager to close any open connection or connection attempt in progress. No further attempts to connect are taken until WF_CMConnect() is called.

Remarks

Disconnection is allowed only in connected state. If MRF24W FW is in the midst of connection (or reconnection) process, then disconnect can hamper connection process, and furthermore it may cause fatal failure in MRF24W FW operation. To be safe to use disconnect, we strongly recommend the user to disable module FW connection manager by setting WF_MODULE_CONNECTION_MANAGER == WF_DISABLED in drv_wifi_config.h

Preconditions

MACInit must be called.

Function

```
uint16_t WF_CMDDisconnect(void)
```

1.9.2.4.1.2 WF_CMInfoGetFSMStats Function**File**

drv_wifi_api.h

Syntax

```
void WF_CMInfoGetFSMStats(tWFCMInfoFSMStats * p_info);
```

Description

This is function WF_CMInfoGetFSMStats.

1.9.2.4.1.3 WF_CMGetConnectContext Function

Retrieves WF connection context for MRF24WG0MA/B

File

drv_wifi_api.h

Syntax

```
void WF_CMGetConnectContext(tWFConnectContext * p_ctx);
```

Returns

None.

Description

Retrieves WF connection context for MRF24WG0MA/B

Remarks

None.

Preconditions

MACInit must be called first.

Function

```
void WF_CMGetConnectContext(tWFConnectContext *p_ctx)
```

1.9.2.4.1.4 WF_DisableModuleConnectionManager Function

When compilation flag WF_MODULE_CONNECTION_MANAGER is set to WF_DISABLED, this will disable MRF24W connection manager.

File

drv_wifi_api.h

Syntax

```
void WF_DisableModuleConnectionManager();
```

Returns

None.

Description

Disable MRF24W connection manager.

Remarks

None.

Preconditions

MACInit must be called first.

Function

```
void WF_DisableModuleConnectionManager(void)
```

1.9.2.5 Wi-Fi Scan

Functions to direct the MRF24WB0M / MRF24WG0M to initiate a site survey

Description

If the application already knows the network SSID that it wants to join than it can set up a connection profile with that information and attempt to join the network. However, there are applications that first need to dynamically determine what infrastructure, adhoc or Wi-Fi Direct networks are in the area, and then decide which network to join. The scan API functions are used to gather this information.

There are 2 types of scan operations

- Active Scan

STA will transmit probe request frames and AP/routers will respond by transmitting probe response frames. For AP/Router with hidden SSID, active scan is used.

- Passive Scan

AP/router will continuously transmit beacon frames for every beacon interval (BI), any STAs may/will receive these beacons and know of existence of this AP/router.

Scanning operation is tied to channel list (MY_DEFAULT_CHANNEL_LIST).

For example,

```
#define MY_DEFAULT_CHANNEL_LIST {1, 3, 6}
```

specifies scanning operation will be conducted in channels 1, 3, 6.

In FCC regional domain, there are channels 1 to 11. And among these channels, channel 1, 6 and 11 are defined as social channels.

1.9.2.5.1 Scan Operation and Scan Results

Scan operation and type of information contained in scan results

Description

Scan operation and scan result

Refer to TCPIPO WiFi EasyConfig Demo for example on how scanning operation is initiated and how scan results are displayed.

Use function prototype `WF_Scan()` to invoke a scan operation.

When the scan results are ready, a `WF_EVENT_SCAN_RESULTS_READY` event will be generated.

Use function prototype `WF_ScanGetResult()` to retrieve scan result from the MRF24WB0M / MRF24WG0M.

Refer to data struct `tWFScanResult` (`drv_wifi_api.h`)

Each scan result returned by MRF24WB0M / MRF24WG0M RF module will contain

- `bssid` (Network BSSID value)
- `ssid` (Network SSID value)
- Access point configuration information such as security mode (WPA2, WPA, WEP, OPEN)

For WEP security mode, the scan results will NOT indicate whether it is `WF_SECURITY_WEP_SHAREDKEY` or `WF_SECURITY_WEP_OPENKEY`. Referring to 802.11 specifications, the beacon itself does not indicate this difference. The scan results reflect whatever information is contained in the beacons or probe responses.

- Access point beacon interval
- `atimWindow` (Applicable only for infrastructure network)
- List of network basic rates and number of valid rates in basic rates
- `RSSI` (Signal strength of received beacon or probe response frames)
- `DtimPeriod`
- `bssType` (`WF_INFRASTRUCTURE`, `WF_ADHOC`)
- `channel` (channel number in which beacon or probe response frame is received in)
- `ssidLen` (Length of SSID)

Remember scan results are retained on the MRF24W until:

1. Calling `WF_Scan()` again (after scan results returned from previous call).
2. Resetting MRF24W.

Scan Results Showing Multiple Copies of the Same SSID

If the same SSID is detected on different channels, the scan results may seem to show duplicate scan results. In reality, the

scan results are showing the same SSID but on different channels. For example, SSID "MCHP_Network" is detected on channel 1, 6 and 11. Therefore the scan results will display among the scan results 3 copies of SSID "MCHP_Network".

AP/Routers With Hidden SSID

If an Access Point uses a hidden SSID, then an active scan must be used (see scanType field in the Connection Algorithm). In active scan, MRF24W will transmit a probe request frame and the AP with the hidden SSID will respond by transmitting a probe response frame.

Scan Cache

If necessary, a scan cache can be created such that scan results are retrieved and stored.

As an example, in TCPIP WiFi EasyConfig Demo in SoftAP mode, upon power on reset, a scan operation is invoked and scan results are retrieved from MRF24WG0M and stored into global array (tWFScanResult preScanResult[50]).

After this, wifi startup in softAP mode is initiated via function prototype WF_Connect(). Within WF_Connect(), there are embedded scan operation, which will cause the scan results to be reset to default values.

1.9.2.5.2 Shorter Scan or Connection Duration

Describes the factors influencing scan or connection duration

Description

Scan or Connection Duration

1. Channel selection

Within FCC domain, there are 11 channels.

Channel 1, 6, 11 overlaps and are designated social channels. The longer the scan channel list, the longer will be the scan and connection duration.

To scan all channel

- #define MY_DEFAULT_CHANNEL_LIST {}

To scan selected channels

- #define MY_DEFAULT_CHANNEL_LIST {1,6,11}

2. Delay timing

Within the stack software, there are numerous delays added in.

These selected delay time may be optimized or changed to use shorter delay time.

Example:

In WFEasyConfigProcess(), when softAP enters Hibernate mode, a delay is executed before exiting Hibernate mode.

DelayMs(50); // SOFTAP_ZEROCONF_SUPPORT. Timing reduced from 200 to 50.

Delay timing was originally set to 200msec and then optimized to 50msec.

The delay timing has not been tested on actual products and are just recommended values.

1.9.2.5.3 Use of macro #define MY_DEFAULT_CHANNEL_LIST

Describes the use of macro #define MY_DEFAULT_CHANNEL_LIST

Description

Macro #define MY_DEFAULT_CHANNEL_LIST

Scanning operation is tied to channel lists (for example, channel 1 to 11 for FCC regional domain).

For example, you can specify

- Scanning to be performed in selected channels such as channel 1, 3 and 5.

```
#define MY_DEFAULT_CHANNEL_LIST {1, 3, 5}
```

- Scanning to be performed in all channels.

```
#define MY_DEFAULT_CHANNEL_LIST {}
```

An exception to the use of MY_DEFAULT_CHANNEL_LIST is when MRF24W network type is CFG_WF_SOFT_AP.

MY_DEFAULT_CHANNEL_LIST will instead indicate the channel the MRF24W softAP will reside in.

For example, MRF24W softAP will reside in social channel 6.

```
#define MY_DEFAULT_CHANNEL_LIST {6}
```

To cater for MRF24W softAP network type, 2 more macros are defined

- #define MY_DEFAULT_CHANNEL_LIST_PRESCAN

Before MRF24W starts up as softAP, MRF24W will first perform a scanning operation. And MY_DEFAULT_CHANNEL_LIST_PRESCAN will indicate the channel list to be scanned before starting up as softAP.

- #define MY_DEFAULT_CHANNEL_LIST_POSTSCAN

When MRF24W softAP is redirected to an infrastructure or any non-softAP network types, MY_DEFAULT_CHANNEL_LIST_POSTSCAN is used to indicate the channel list to be scanned in these non-softAP network types.

1.9.2.5.4 Maximum Scan Results

Describes the maximum scan results supported by MRF24W RF modules

Description

Maximum Scan Results Supported

Both MRF24WB0M and MRF24WG0M support up to a maximum of 60 scan results.

1.9.2.5.5 Scan Public Members

Functions and variables accessible by the stack application

Structures

Name	Description
tWFScanResult	Scan Results

Description

The following functions and variables are available to the stack application.

1.9.2.5.5.1 tWFScanResult Structure

File

drv_wifi_api.h

Syntax

```
typedef struct {
    uint8_t bssid[WF_BSSID_LENGTH];
    uint8_t ssid[WF_MAX_SSID_LENGTH];
    uint8_t apConfig;
    uint8_t reserved;
    uint16_t beaconPeriod;
    uint16_t atimWindow;
    uint8_t basicRateSet[WF_MAX_NUM_RATES];
    uint8_t rssi;
    uint8_t numRates;
    uint8_t DtimPeriod;
    uint8_t bssType;
    uint8_t channel;
    uint8_t ssidLen;
} tWFScanResult;
```

Members

Members	Description
uint8_t bssid[WF_BSSID_LENGTH];	Network BSSID value
uint8_t ssid[WF_MAX_SSID_LENGTH];	Network SSID value
uint8_t apConfig;	Access point configuration
uint16_t beaconPeriod;	Network beacon interval
uint16_t atimWindow;	Only valid if bssType = WF_INFRASTRUCTURE
uint8_t basicRateSet[WF_MAX_NUM_RATES];	List of Network basic rates. Each rate has the following format: Bit 7 <ul style="list-style-type: none">0 – rate is not part of the basic rates set1 – rate is part of the basic rates set Bits 6:0 Multiple of 500kbps giving the supported rate. For example, a value of 2 (2 * 500kbps) indicates that 1mbps is a supported rate. A value of 4 in this field indicates a 2mbps rate (4 * 500kbps).
uint8_t rssi;	Signal strength of received frame beacon or probe response
uint8_t numRates;	Number of valid rates in basicRates
uint8_t DtimPeriod;	Part of TIM element
uint8_t bssType;	WF_INFRASTRUCTURE or WF_ADHOC
uint8_t channel;	Channel number
uint8_t ssidLen;	Number of valid characters in ssid

Description

Scan Results

1.9.2.6 Wi-Fi Security

Describes the Wi-Fi security modes supported by MRF24WB0M and MRF24WG0M

Description

This section elaborates on the various security modes supported by MRF24WB0M and MRF24WG0M.

Security modes supported in

- Ad-hoc

OPEN, WEP40/104

- SoftAP

OPEN, WEP40/104

- Infrastructure

OPEN, WEP40/104, WPA-PSK/WPA2-PSK, WPS PBC/PIN

- Wi-Fi Direct

WPS

1.9.2.6.1 Wired Equivalent Privacy (WEP)

Describes the Wi-Fi security modes Wired Equivalent Privacy (WEP)

Description

WIRED EQUIVALENT PRIVACY (WEP)

Security loop holes are present and this is not recommend!!!

WEP security modes supported are

- WF_SECURITY_WEP_40

WEP Encryption using 40 bit keys. Also known as WEP 64 keys.

This security method uses a 40 bit (10 Hex character) "secret key" and a 24 bit Initialization Vector (IV).

- WF_SECURITY_WEP_104

WEP Encryption using 104 bit keys Also known as WEP 128 keys.

This security method uses a 104 bit (26 Hex Character) "secret key", and a 24 bit Initialization Vector (IV).

MRF24W only accepts WEP hex keys (MY_DEFAULT_WEP_KEYS_40 or MY_DEFAULT_WEP_KEYS_104) and NOT passphrase. Some Web sites offer this automatic WEP key generators, whereby WEP keys are generated from ordinary text called a passphrase.

WEP key types supported are

- WF_SECURITY_WEP_OPENKEY

Default

Both AP/router & client STA do not use the key during the connection process (authentication and association). Thus the connection process is exactly the same as open mode. Once the connection is established, then both AP/router and client STA can start to use the key to encrypt the data packets. For the case when the key mismatches, connection can still be established. However, it will fail during operations such as DHCP, etc.

- WF_SECURITY_WEP_SHAREDKEY

Supported by MRF24WG0M.

Supported by MRF24WB0M (RF module FW version 0x1209 and later)

This involves the key during the authentication process. When the key mismatches, this will cause the connection process to fail.

According to 802.11 specifications, WEP can have a total of 4 keys (or 4 key indices). However, in commercial products , only 1 key index (0) is really used. As an example, for iOS, Android and even Windows, there is probably no option to choose WEP key index. That implies they are using only 1 WEP key index.

1.9.2.6.2 Wi-Fi Protected Access (WPA/WPA2)

Describes the Wi-Fi security modes Wi-Fi Protected Access (WPA/WPA2)

Description

Wi-Fi PROTECTED ACCESS (WPA/WPA2)

Refer to the Wi-Fi Protected Access (WPA) Enhanced Security Implementation Based on IEEE P802.11i standard.

Other equivalent WPA/WPA2 terminologies used are WPA - 4 way handshake or EAPOL - 4 way handshake or 802.1X authentication.

Upon initial connection, after authentication followed by association, WPA/WPA2 EAPOL 4-way handshaking will takes place.

WPA/WPA2 security modes supported are

- `WF_SECURITY_WPA_AUTO_WITH_KEY`

WPA-PSK Personal or WPA2-PSK Personal where binary key is given and MRF24W will connect at highest level AP supports (WPA-PSK/WPA2-PSK)

- `WF_SECURITY_WPA_AUTO_WITH_PASS_PHRASE`

WPA-PSK Personal or WPA2-PSK Personal where passphrase is given to MRF24W and it calculates the binary key and connects at highest level AP supports (WPA-PSK/WPA2-PSK).

MRF24W can be configured to accept either binary key or passphrase, through `MY_DEFAULT_WIFI_SECURITY_MODE` compile-time definiton. The WPA/WPA2 authentication process involves key derivation from the given passphrase.

Since this key derivation process is computational intensive and requires memory space, options are given

- **Host to derive the key from the passphrase**

`DERIVE_KEY_FROM_PASSPHRASE_IN_HOST` needs to be enabled. Refer to function prototype `pbkdf2_sha1()` in `drv_wifi_pbkdf2.c` for the algorithm.

In this scenario, MRF24W will generate an event (`WF_EVENT_KEY_CALCULATION_REQUEST`) to host (eg PIC32) and set `g_WpsPassphrase.valid` to `TRUE`. Upon receipt of this event, the host will execute the function `WF_ConvPassphrase2Key()` to convert the passphrase to key. Upon completion of this conversion, the host will call `WF_SetPSK()` to pass the converted key to MRF24W.

- **MRF24W will handle the entire WPA/WPA2 passphrase and then key derivation**

MRF24WB0M consumes about ~32 seconds whereas MRF24WG0M will consume about ~25 seconds.

1.9.2.6.3 Wi-Fi Protected Setup (WPS)

Describes the Wi-Fi security connection modes Wi-Fi Protected Setup (WPS). WPS security mode is only supported by MLA v5.42 July 2012 releases or later. Only MRF24WG0M supports WPS security mode.

Structures

Name	Description
tWFWpsCred	This is type tWFWpsCred.

Description**Wi-Fi PROTECTED SETUP (WPS)**

Refer to the Wi-Fi Protected Setup Specification Version 1.0h standard.

WiFi Protected Setup (WPS) allows users to set up and expand the WiFi networks with security enabled, even if they are not familiar with the underlying technologies or processes involved. For example, users no longer have to know that SSID refers to the network name or WPA2 refers to the security mechanism.

WPS does not support ad-hoc networks.

WPS will configure the network name SSID and security key for the AP and WPS client devices on a network. It supports the WEP / WPA-PSK / WPA2-PSK security methods.

2 methods are supported

- WPS-PBC (Push Button Configuration)

Users can connect the device (MRF24WG0M) to the network and enable data encryption by pushing the buttons on the AP and client device. Users do NOT need to know the SSID of the AP, however the users are required to be within close proximity of the AP to press the push button on the AP.

- WPS-PIN (Personal Information Number)

PIN is provided for each device which joins the network. Enter this PIN on the AP/Router (Registrar) and activate AP (Registrar) first before MRF24WG0M attempts to connect. The last digit of the PIN is the checksum of the first 7 digits of the PIN. This checksum must be correct, otherwise MRF24WG0M module will reject the PIN code.

To set up WPS-PBC, define

- MY_DEFAULT_NETWORK_TYPE as CFG_WF_INFRASTRUCTURE
- MY_DEFAULT_SSID_NAME as ""
- MY_DEFAULT_WIFI_SECURITY_MODE as WF_SECURITY_WPS_PUSH_BUTTON

To set up WPS-PIN, define

- MY_DEFAULT_NETWORK_TYPE as CFG_WF_INFRASTRUCTURE
- MY_DEFAULT_SSID_NAME to be the same SSSID as the AP/router
- MY_DEFAULT_WIFI_SECURITY_MODE as WF_SECURITY_WPS_PIN
- MY_DEFAULT_WPS_PIN to be the same as the AP/router PIN.

WPS protocol can be viewed as a security connection method, built upon the existing security modes WPA-PSK/WPA2-PSK. The protocol encompasses a M1-M8 message exchange process. Therefore additional time is consumed. The WPS specification specifies below:

1. Retransmission timeout = 5 sec
2. Individual message processing timeout = 15 sec
3. Overall timeout for entire protocol to complete = 2 min

To address this lengthy time required every single time the MRF24W is restarted, the function prototype `WF_SaveWPSCredentials()` can be invoked. Basically what this function does is to retrieve WPS credentials from MRF24W and store these into global variable `AppConfig`. In this way, it makes the WPS credentials re-useable and shorten subsequent reconnection time. For an actual product, it is advised to add in a timeout such that the WPS credentials will need to be refreshed or updated periodically.

WPS protocol

The WPS protocol, which encompasses WPA-PSK/WPA2-PSK authentication process, involves key derivation from the given passphrase. Since this key derivation process is computational intensive and requires memory space, options are given

- **Host to derive the key from the passphrase**

DERIVE_KEY_FROM_PASSPHRASE_IN_HOST needs to be enabled. Refer to function prototype pbkdf2_sha1() in drv_wifi_pbkdf2.c for the algorithm.

In this scenario, WF_YieldPassphrase2Host() will inform MRF24W that host wants to do conversion. MRF24W will generate an event (WF_EVENT_KEY_CALCULATION_REQUEST) to host (eg PIC32) and set g_WpsPassphrase.valid to TRUE. Upon receipt of this event, the host will execute the function WF_ConvPassphrase2Key() to convert the passphrase to key. Upon completion of this conversion, the host will call WF_SetPSK() to pass the converted key to MRF24W.

- **MRF24W will handle the entire WPA-PSK/WPA2-PSK passphrase and then key derivation**

MRF24WG0M will consume about ~25 seconds.

Upon initial connection (after authentication followed by association), WPS will take place and is performed over 2 phases.

Phase A: WPS Frame Exchanges (EAP authentication protocol)

Refer to Wi-Fi Simple Configuration Specification Version 2.0.2 or Wi-Fi Protected Setup Specification Version 1.0h "Figure 4: In-band Setup Using a Standalone AP/Registrar".

Information and network credentials, such as SSID, security mode, security keys, security passphrase, etc, are securely exchanged over the air using the Extensible Authentication Protocol (EAP). From these WPS frame exchanges, WPS will automatically configure the network connection, without having the user to know the SSID and security keys or passphrases, etc.

The enrollee is defined as a device seeking to join a wireless network and is represented by MRF24WG0M. In the infrastructure network, the enrollee and wireless client are synonymous.

WSC IE (Wi-Fi Simple Configuration Information Element) will be present in the beacons, probe request/responses frames and association request/response frames. Refer to Wi-Fi Simple Configuration Specification for details on the WSC IE contents.

Phase B: EAPOL 4-way Handshake or 802.1X-authentication

Refer to specifications IEEE 802.11i-2004: Amendment 6: Medium Access Control (MAC) Security Enhancements.

After a successful EAP authentication, the EAPOL 4-way handshake begins.

The supplicant, STA and wireless client are synonymous.

Likewise, the authenticator and AP/router are synonymous.

The 4-way handshake shares unique random information between the supplicant/client and the authenticator /AP to derive the PTK key.

Below is a brief description of the EAPOL 4-way handshake

4-way handshake message 1

The AP/router sends the STA a nonce (ANonce). Along with this ANonce, the frame includes the AP/router MAC address. At this point the STA has all the information needed to create the PTK; ANonce, AP/router MAC address, its own Snonce and MAC address.

4-way handshake message 2

The STA creates its nonce (SNonce). The STA can now calculate the PTK because it has all the information from the first handshake. In this second message, the STA sends the SNonce to the AP/router. The STA also sends the security

parameters (RSN) information. The entire message gets an authentication check using the (MIC) from the pairwise key hierarchy. The AP/router can then verify that the information, including the security parameters sent at association is valid.

4-way handshake message 3

In this third message, the AP/router derives the GTK key from the GMK key. The AP/router derives an ANonce, RSN information element info and a MIC and sends these information to the STA in an EAPOL-Key frame. This is kept secret from sniffing since it is encrypted within the PTK.

4-way handshake message 4

The fourth message acts as a confirmation. It indicates that the temporal keys are installed.

Below shows the WPS protocol

Integration of WPS into 802.11 joining operation

Below lists the overall sequences

- Scanning, Authentication, Association
 - WPS Frame Exchanges (EAP protocol)
 - Deauthentication (Refer to above figure) or Disassociation
 - o Some APs are found to transmit disassociation instead of deauthentication frame.
- Provision needs to be made to handle receipt of disassociation frame.
- Authentication, Association
 - EAPOL 4-way handshake or 802.1X-authentication

1.9.2.6.3.1 tWFWpsCred Structure

File

drv_wifi_api.h

Syntax

```
typedef struct {
    uint8_t ssid[32];
    uint8_t netKey[64];
    uint16_t authType;
    uint16_t encType;
    uint8_t netIdx;
    uint8_t ssidLen;
    uint8_t keyIdx;
    uint8_t keyLen;
    uint8_t bssid[6];
} tWFWpsCred;
```

Members

Members	Description
uint8_t ssid[32];	SSID
uint8_t netKey[64];	Net Key PSK
uint16_t authType;	Authentication Type: AUTH_OPEN / AUTH_WPA_PSK / AUTH_SHARED / AUTH_WPA / AUTH_WPA2 / AUTH_WPA2_PSK
uint16_t encType;	Encoding Type: ENC_NONE / ENC_WEP / ENC_TKIP / ENC_AES
uint8_t netIdx;	Net ID
uint8_t ssidLen;	SSID length
uint8_t keyIdx;	Key ID

uint8_t keyLen;	WPA-PSK/WPA2-PSK key length
uint8_t bssid[6];	BSSID

Description

This is type tWFWpsCred.

1.9.2.7 Wi-Fi Tx Power Control

API to control the transmit (Tx) power of the MRF24WB0M / MRF24WG0M

Description

The API functions in this section are used to configure the MRF24WB0M / MRF24WG0M transmit (Tx) power control settings.

MRF24WB0M transmit power settings are from -10dBm to +10dBm.

MRF24WG0M transmit power settings are from 0dBm to +18dBm.

1.9.2.7.1 Tx Power Control Public Members

Functions and variables accessible by the stack application

Description

The following functions and variables are available to the stack application.

1.9.2.8 Wi-Fi Power Save

Functions to alter the power savings features of the MRF24WB0M / MRF24WG0M

Description

The MRF24WB0M / MRF24WG0M supports two power-saving modes – sleep and hibernate.

Mode	Description
Sleep	<p>This mode is used when in PS Poll mode where the MRF24WB0M / MRF24WG0M wakes itself up at periodic intervals to query the network for receive messages buffered by an Access Point. See listenInterval in the tWFCACElements structure.</p> <p>When in sleep mode the MRF24WB0M / MRF24WG0M transmitter receiver circuits are turned off along with other circuitry to minimize power consumption.</p> <p>Sleep mode is entered periodically as a result of the Host CPU enabling PS Poll mode.</p>
Hibernate	<p>This mode effectively turns off the LDO of the MRF24WB0M / MRF24WG0M for maximum power savings. MRF24WB0M / MRF24WG0M state is not retained, and when the MRF24WB0M / MRF24WG0M is taken out of the Hibernate state it performs a reboot.</p> <p>Hibernate mode is controlled by toggling the HIBERNATE pin on the MRF24WB0M / MRF24WG0M module (high to enter hibernate, low to exit).</p> <p>This mode should be used when the application allows for the MRF24WB0M / MRF24WG0M module to be off for extended periods of time.</p>

Power Save Functions

802.11 chipsets have two well known operational power modes. Active power mode is defined as the radio always on either transmitting or receiving, meaning that when it isn't transmitting then it is trying to receive. Power save mode is defined as operating with the radio turned off when there is nothing to transmit and only turning the radio receiver on when required.

The power save mode is a mode that requires interaction with an Access Point. The access point is notified via a packet from the Station that it is entering into power save mode. As a result the access point is required to buffer any packets that are destined for the Station until the Station announces that it is ready to once again receive packets. The duration that a

Station is allowed to remain in this mode is limited and is typically 10 times the beacon interval of the Access point.



If the host is expecting packets from the network it should operate in Active mode. If however power saving is critical and packets are not expected then the host should consider operating in power save mode. Due to the nature of Access points not all behaving the same, there is the possibility that an Access point will invalidate a Stations connection if it has not heard from the Station over a given time period. For this reason power save mode should be used with caution.

The 802.11 name for power saving mode is PS-Poll (Power-Save Poll).

1.9.2.8.1 Power Save Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
	WF_HibernateEnable	Puts the MRF24W into hibernate mode by setting HIBERNATE pin to HIGH.
	WF_PsPollDisable	Disables PS-Poll mode.

Structures

	Name	Description
	WFHibernate	This is record WFHibernate.

Description

The following functions and variables are available to the stack application.

1.9.2.8.1.1 WF_HibernateEnable Function

Puts the MRF24W into hibernate mode by setting HIBERNATE pin to HIGH.

File

drv_wifi_api.h

Syntax

```
void WF_HibernateEnable( );
```

Returns

None.

Description

Enables Hibernate mode on the MRF24W, which effectively turns off the device for maximum power savings. HIBERNATE pin on MRF24W is set to HIGH.

MRF24W state is not maintained when it transitions to hibernate mode. To remove the MRF24W from hibernate mode call WF_Init().

Remarks

Note that because the MRF24W does not save state, there will be a disconnect between the TCP/IP stack and the MRF24B0M state. If it is desired by the application to use hibernate, additional measures must be taken to save application state. Then the host should be reset. This will ensure a clean connection between MRF24W and TCP/IP stack

Future versions of the stack might have the ability to save stack context as well, ensuring a clean wake up for the MRF24W without needing a host reset.

Preconditions

MACInit must be called first.

Function

```
void WF_HibernateEnable()
```

1.9.2.8.1.2 WF_PsPollDisable Function

Disables PS-Poll mode.

File

drv_wifi_api.h

Syntax

```
void WF_PsPollDisable();
```

Returns

None.

Description

Disables PS Poll mode. The MRF24W will stay active and not go sleep.

Remarks

None.

Preconditions

MACInit must be called first.

Function

```
void WF_PsPollDisable(void)
```

1.9.2.8.1.3 WFHibernate Structure

File

drv_wifi_api.h

Syntax

```
struct WFHibernate {  
    uint8_t state;  
    uint8_t wakeup_notice;  
};
```

Description

This is record WFHibernate.

1.9.2.8.2 Power Save Internal Members

Functions and variables internal to the module

Description

The following functions and variables are designated as internal to the module.

1.9.2.9 Wi-Fi Process Event

Describes how to receive and act on events from the MRF24WB0M / MRF24WG0M

Description

\

There are several events that can occur on the MRF24WB0M / MRF24WG0M that the host CPU may want to know about. All MRF24WB0M / MRF24WG0M events go through the `WF_ProcessEvent()` function described in the next section.

Event Processing

The `WF_ProcessEvent()` function is how the host application is notified of events. This function will be called by the Wi-Fi host driver when an event occurs. This function should not be called directly by the host application. This function, located in `WF_Config.c`, should be modified by the user as needed. Since this function is called from the Wi-Fi driver there are some restrictions – namely, one cannot call any Wi-Fi driver functions when inside `WF_ProcessEvent()`. It is recommended that that customer simply set a flag for a specific event and handle it in the main loop. The framework for this function is shown below.

The prototype for this function is:

```
void WF_ProcessEvent(UINT8 event, UINT16 eventInfo, UINT8 *extraInfo)
```

There are 3 inputs to the function:

event	The event that occurred.
eventInfo	Additional information about the event. Not all events have associated info, in which case this value will be set to <code>WF_NO_ADDITIONAL_INFO</code> (0xff)
*extraInfo	Additional information about the event. When <code>DERIVE_KEY_FROM_PASSPHRASE_IN_HOST</code> is enabled, where host will compute the key from the passphrase, this field contains the WPA Passphrase that will be sent to the host for the computation.

The table below shows possible values that the event and eventInfo parameters can have. Note that event notification of some events can be optionally disabled via:

1. Bit mask `eventNotificationAction` in the `tWFCAElements` structure (see Wi-Fi Connection Algorithm), or
2. Function `WF_CASetEventNotificationAction()`.

event	eventInfo
<code>WF_EVENT_CONNECTION_SUCCESSFUL</code>	The connection attempt was successful. eventInfo: <ul style="list-style-type: none"> • Always <code>WF_NO_ADDITIONAL_INFO</code> (Optional event)
<code>WF_EVENT_CONNECTION_FAILED</code>	The connection attempt failed eventInfo: <ul style="list-style-type: none"> • <code>WF_JOIN_FAILURE</code> • <code>WF_AUTHENTICATION_FAILURE</code> • <code>WF_ASSOCIATION_FAILURE</code> • <code>WF_WEP_HANDSHAKE_FAILURE</code> • <code>WF_PSK_CALCULATION_FAILURE</code> • <code>WF_PSK_HANDSHAKE_FAILURE</code> • <code>WF_ADHOC_JOIN_FAILURE</code> • <code>WF_SECURITY_MISMATCH_FAILURE</code> • <code>WF_NO_SUITABLE_AP_FOUND_FAILURE</code> • <code>WF_RETRY_FOREVER_NOT_SUPPORTED_FAILURE</code> (Optional event)

WF_EVENT_CONNECTION_TEMPORARILY_LOST	<p>An established connection was temporarily lost – the connection algorithm is attempting to reconnect. The eventInfo field indicates why the connection was lost.</p> <p>eventInfo:</p> <ul style="list-style-type: none"> WF_BEACON_TIMEOUT WF_DEAUTH_RECEIVED WF_DISASSOCIATE_RECEIVED <p>(Optional event)</p>
WF_EVENT_CONNECTION_PERMANENTLY_LOST	<p>An established connection was permanently lost – the connection algorithm either ran out of retries or was configured not to retry. The eventInfo field indicates why the connection was lost.</p> <p>eventInfo:</p> <ul style="list-style-type: none"> WF_BEACON_TIMEOUT WF_DEAUTH_RECEIVED WF_DISASSOCIATE_RECEIVED <p>This event can also be generated when WF_CMDDisconnect() is called, in which case the eventInfo field has no meaning.</p> <p>(Optional event)</p>
WF_EVENT_CONNECTION_REESTABLISHED	<p>A connection that was temporarily lost has been reestablished</p> <p>Always WF_NO_ADDITIONAL_INFO</p> <p>(Optional event)</p>
WF_EVENT_SCAN_RESULTS_READY	<p>The scan request initiated by calling WF_Scan() has completed and results can be read from the MRF24WB0M / MRF24WG0M.</p> <p>eventInfo: Number of scan results</p>
WF_EVENT_SOFT_AP_EVENT	<p>Available only for MRF24WG0M (i) FW version 0x3108 and later and (ii) MLA v5.42.06 release or later. Indication of client's connection status, when a client has connected or disconnected or not powered on/active or received deauthentication.</p>
WF_EVENT_KEY_CALCULATION_REQUEST	<p>This event is generated when DERIVE_KEY_FROM_PASSPHRASE_IN_HOST is enabled, MRF24WG0M will transmit the passphrase to the host via the field *extraInfo. where the host will then compute the passphrase from the key. Refer to function prototype WF_ConvPassphrase2Key() for more information.</p>

12.2 WF_ProcessEvent() Framework

Below is the framework for WF_ProcessEvent(). Each case statement should be modified as needed to handle events the application is interested in.

```
void WF_ProcessEvent(UINT8 event, UINT16 eventInfo)
{
    switch (event)
    {
        case WF_EVENT_CONNECTION_SUCCESSFUL:
            /* Application code here */
            break;

        case WF_EVENT_CONNECTION_FAILED:
            /* Application code here */
            break;

        case WF_EVENT_CONNECTION_TEMPORARILY_LOST:
            /* Application code here */
            break;
    }
}
```

```

        break;

    case WF_EVENT_CONNECTION_PERMANENTLY_LOST:
        /* Application code here */
        break;
    case WF_EVENT_CONNECTION_REESTABLISHED:
        /* Application code here */
        break;
    case WF_EVENT_SCAN_RESULTS_READY:
        /* Application code here */
        break;

    default:
        WF_ASSERT(FALSE);
        break;
}

```

1.9.2.9.1 WiFi MRF24WG Events

Summary

EventInfo - Upper byte indicates Status. Lower byte indicates Reason.

Description

Status Code = (uint8) (eventinfo >> 8); Reason Code = (uint8)(eventinfo & 0xff);

Status Code	Reason Code
0	"" - not used
1	"" - not used
2	WF_JOIN_FAILURE
3	WF_AUTHENTICATION_FAILURE
4	WF_ASSOCIATION_FAILURE
5	WF_WEP_HANDSHAKE_FAILURE
6	WF_PSK_CALCULATION_FAILURE
7	WF_PSK_HANDSHAKE_FAILURE"
8	WF_ADHOC_JOIN_FAILURE"
9	WF_SECURITY_MISMATCH_FAILURE
10	WF_NO_SUITABLE_AP_FOUND_FAILURE
11	WF_RETRY_FOREVER_NOT_SUPPORTED_FAILURE
12	WF_LINK_LOST
13	"" - not used
14	WF_RSN_MIXED_MODE_NOT_SUPPORTED
15	WF_RECV_DEAUTH
16	WF_RECV_DISASSOC
17	WF_WPS_FAILURE
18	WF_P2P_FAILURE
19	WF_LINK_DOWN

Description

Reason varies per Status. When Status is either 15(WF_RECV_DEAUTH) or 16 (WF_RECV_DISASSOC), Reason tells De-Auth or Dis-Assoc reason.

Reason Code	Description
0	"" - not used
1	WF_UNSPECIFIED
2	WF_PREV_AUTH_NOT_VALID
3	WF_DEAUTH_LEAVING
4	WF_DISASSOC_DUE_TO_INACTIVITY
5	WF_DISASSOC_AP_BUSY
6	WF_CLASS2_FRAME_FROM_NONAUTH_STA
7	WF_CLASS3_FRAME_FROM_NONASSOC_STA
8	WF_DISASSOC_STA_HAS_LEFT
9	WF_STA_REQ_ASSOC_WITHOUT_AUTH
10	"" - not used
11	"" - not used
12	"" - not used
13	WF_INVALID_IE
14	WF_MIC_FAILURE
15	WF_4WAY_HANDSHAKE_TIMEOUT
16	WF_GROUP_KEY_HANDSHAKE_TIMEOUT
17	WF_IE_DIFFERENT
18	WF_INVALID_GROUP_CIPHER
19	WF_INVALID_PAIRWISE_CIPHER
20	WF_INVALID_AKMP
21	WF_UNSUPP_RSN_VERSION
22	WF_INVALID_RSN_IE_CAP
23	WF_IEEE8021X_FAILED
24	WF_CIPHER_SUITE_REJECTED

Description

When Status is either 3 (WF_AUTHENTICATION_FAILURE) or 4 (WF_ASSOCIATION_FAILURE), Reason tells failure reason.

Reason Code	Description
0	"" - not used
1	WF_UNSPECIFIED_FAILURE
2	"" - not used
3	"" - not used
4	"" - not used

5	"" - not used
6	"" - not used
7	"" - not used
8	"" - not used
9	"" - not used
10	WF_CAPS_UNSUPPORTED
11	WF_REASSOC_NO_ASSOC
12	WF_ASSOC_DENIED_UNSPEC
13	WF_NOT_SUPPORTED_AUTH_ALG
14	WF_UNKNOWN_AUTH_TRANSACTION
15	WF_CHALLENGE_FAIL
16	WF_AUTH_TIMEOUT
17	WF_AP_UNABLE_TO_HANDLE_NEW_STA
18	WF_ASSOC_DENIED_RATES
19	WF_ASSOC_DENIED_NOSHORTPREAMBLE
20	WF_ASSOC_DENIED_NOPBCC
21	WF_ASSOC_DENIED_NOAGILITY
22	"" - not used
23	"" - not used
24	"" - not used
25	WF_ASSOC_DENIED_NOSHORTTIME
26	WF_ASSOC_DENIED_NODSSSOFTDM
27	"" - not used
28	"" - not used
29	"" - not used
30	"" - not used
31	"" - not used
32	"" - not used
33	"" - not used
34	"" - not used
35	"" - not used
36	"" - not used
37	"" - not used
38	"" - not used
39	"" - not used
40	WF_NOT_VALID_IE
41	WF_NOT_VALID_GROUPCIPHER
42	WF_NOT_VALID_PAIRWISE_CIPHER
43	WF_NOT_VALID_AKMP
44	WF_UNSUPPORTED_RSN_VERSION
45	WF_INVALID_RSN_IE_CAP
46	WF_CIPHER_SUITE_REJECTED

47	WF_TIMEOUT
----	------------

Description

When status is 17 (WF_WPS_FAILURE), Reason code is divided to WpsState (upper 4 bits) and WpsConfigError (lower 4 bits) WpsState = reason >> 4; WpsConfigError = reason & 0x0f;

WpsState	Description
0	"NONE"
1	"EAPOL_START"
2	"EAP_REQ_IDENTITY"
3	"EAP_RSP_IDENTITY"
4	"EAP_WPS_START"
5	"EAP_RSP_M1"
6	"EAP_REQ_M2"
7	"EAP_RSP_M3"
8	"EAP_REQ_M4"
9	"EAP_RSP_M5"
10	"EAP_REQ_M6"
11	"EAP_RSP_M7"
12	"EAP_REQ_M8"
13	"EAP_RSP_DONE"
14	"EAP_FAILURE"

WpsConfigError	Description
0	NOERR
1	"SESSION_OVERLAPPED"
2	"DECRYPT_CRC_FAILURE"
3	"24G_NOT_SUPPORTED"
4	"RETRY_FAILURE"
5	"INVALID_MSG"
6	"AUTH_FAILURE"
7	"ASSOC_FAILURE"
8	"MSG_TIMEOUT"
9	"SESSION_TIMEOUT"
10	"DEVPASSWD_AUTH_FAILURE"
11	"NO_CONN_TOREG"
12	"MULTI_PBC_DETECTED"
13	"EAP_FAILURE"
14	"DEV_BUSY"
15	"SETUP_LOCKED"

Description

When Status is 18 (WF_P2P_FAILURE), Reason code is divided to P2PState (upper 4 bits) and P2PError (lower 4 bits).
P2PState = reason >> 4; P2PError = reason & 0x0f;

P2PState	Description
0	"WFD_SUCCESS"
1	"WFD_INFO_CURRENTLY_UNAVAILABLE"
2	"WFD_INCOMPATIBLE_PARAMS"
3	"WFD_LIMIT_REACHED"
4	"WFD_INVALID_PARAMS"
5	"WFD_UNABLE_TO_ACCOMMODATE"
6	"WFD_PREV_PROTOCOL_ERROR"
7	"WFD_NO_COMMON_CHANNELS"
8	"WFD_UNKNOWN_GROUP"
9	""
10	"WFD_INCOMPATIBLE_PROV_METHOD"
11	"WFD_REJECTED_BY_USER"
12	"WFD_NO_MEM"
13	"WFD_INVALID_ACTION"
14	"WFD_TX_FAILURE"
15	"WFD_TIME_OUT"

P2PError	Description
0	"P2PSTIdle"
1	"P2PSTScan"
2	"P2PSTListen"
3	"P2PSTFind"
4	"P2PSTStartFormation"
5	"P2PSTGONegoReqDone"
6	"P2PSTGOWaitNegoReqDone"
7	"P2PSTWaitFormationDone"
8	"P2PSTInvite"
9	"P2PSTProvision"
10	"P2PSTClient"

1.9.2.10 Wi-Fi Miscellaneous





Functions for controlling miscellaneous features of the MRF24WB0M / MRF24WG0M

Description




1.9.2.10.1 Wi-Fi Miscellaneous Public Members

Functions and variables accessible by the stack application

Functions

	Name	Description
	WF_GetDeviceInfo	Retrieves WF device information (MRF24WB0M_DEVICE/MRF24WG0M_DEVICE, romVersion and patchVersion).
	WF_GetMacStats	Gets MAC statistics.
	WF_EnableSWMultiCastFilter	Forces the module FW to use software filter instead of hardware filter
	WF_MulticastSetConfig	Sets a multicast address filter using one of the two multicast filters.

Structures

	Name	Description
	WFMacStatsStruct	Used in WF_GetMacStats.
	WFMulticastConfigStruct	Used in WF_MulticastSetConfig, WF_MulticastGetConfig.
	tWFDeviceInfoStruct	used in WF_GetDeviceInfo

Description

The following functions and variables are available to the stack application.

1.9.2.10.1.1 WF_GetDeviceInfo Function

Retrieves WF device information (MRF24WB0M_DEVICE/MRF24WG0M_DEVICE, romVersion and patchVersion).

File

drv_wifi_api.h

Syntax

```
void WF_GetDeviceInfo(tWFDeviceInfo * p_deviceInfo);
```

Returns

None.

Description

Retrieves RF module information.

- MRF24WB will have romVersion = 0x12.
- MRF24WG will have romVersion = 0x30 or 0x31.

Remarks

None.

Preconditions

MACInit must be called first.

Parameters

Parameters	Description
tWFDeviceInfo * p_deviceInfo	Pointer where device info will be written

Function

```
void WF_GetDeviceInfo(tWFDeviceInfo *p_deviceInfo)
```

1.9.2.10.1.2 WF_GetMacStats Function

Gets MAC statistics.

File

drv_wifi_api.h

Syntax

```
void WF_GetMacStats(tWFMacStats * p_macStats);
```

Returns

None.

Description

Returns MAC statistics on number of frames received or transmitted for defined situations such as number of frames transmitted with multicast bit set in destination MAC address. Refer to drv_wifi_api.h for data struct WFMacStatsStruct / tWFMacStats.

Remarks

None.

Preconditions

MACInit must be called first.

Parameters

Parameters	Description
tWFMacStats * p_macStats	Pointer to where MAC statistics are written

Function

```
void WF_GetMacStats(tWFMacStats *p_macStats)
```

1.9.2.10.1.3 WF_EnableSWMultiCastFilter Function

Forces the module FW to use software filter instead of hardware filter

File

drv_wifi_api.h

Syntax

```
void WF_EnableSWMultiCastFilter();
```

Returns

None.

Description

This function allows the application to configure up to 16 software-based Multicast Address Filters on the MRF24WG0MA/B.

Remarks

Definition WF_SOFTWARE_MULTICAST_FILTER needs to be set to WF_ENABLED

Preconditions

MACInit must be called first.

Function

```
void WF_EnableSWMultiCastFilter(void)
```

1.9.2.10.1.4 WF_MulticastSetConfig Function

Sets a multicast address filter using one of the two multicast filters.

File

drv_wifi_api.h

Syntax

```
void WF_MulticastSetConfig(tWFMultiCastConfig * p_config);
```

Returns

None.

Description

This function allows the application to configure up to two Multicast Address Filters on the MRF24W. If two active multicast filters are set up they are OR'd together – the MRF24W will receive and pass to the Host CPU received packets from either multicast address. The allowable values in p_config (tWFMultiCastConfig / WFMulticastConfigStruct) are:

- filterId -- WF_MULTICAST_FILTER_1 or WF_MULTICAST_FILTER_2
- action -- WF_MULTICAST_DISABLE_ALL (default) The Multicast Filter discards all received multicast messages – they will not be forwarded to the Host PIC. The remaining fields in this structure are ignored.

WF_MULTICAST_ENABLE_ALL The Multicast Filter forwards all received multicast messages to the Host PIC. The remaining fields in this structure are ignored.

WF_MULTICAST_USE_FILTERS The MAC filter will be used and the remaining fields in this structure configure which Multicast messages are forwarded to the Host PIC.

- macBytes -- Array containing the MAC address to filter on (using the destination address of each incoming 802.11 frame). Specific bytes with the MAC address can be designated as 'don't care' bytes. See macBitMask. This field is only used if action = WF_MULTICAST_USE_FILTERS.
- macBitMask -- A byte where bits 5:0 correspond to macBytes[5:0]. If the bit is zero then the corresponding MAC byte must be an exact match for the frame to be forwarded to the Host PIC. If the bit is one then the corresponding MAC byte is a 'don't care' and not used in the Multicast filtering process. This field is only used if action = WF_MULTICAST_USE_FILTERS.

By default, both Multicast Filters are inactive.

Remarks

Definition WF_USE_MULTICAST_FUNCTIONS needs to be enabled.

Preconditions

MACInit must be called first.

Example

- Filter on Multicast Address of 01:00:5e:xx:xx:xx where xx are don't care bytes. p_config->filterId = WF_MULTICAST_FILTER_1

[0] [1] [2] [3] [4] [5] p_config->macBytes[] = 01, 00, 5e, ff, ff, ff (0xff are the don't care bytes)

p_config->macBitMask = 0x38 --> bits 5:3 = 1 (don't care on bytes 3,4,5) --> bits 2:0 = 0 (exact match required on bytes 0,1,2)

Function

```
void WF_MulticastSetConfig(tWFMultiCastConfig *p_config);
```

1.9.2.10.1.5 WFMacStatsStruct Structure

File

drv_wifi_api.h

Syntax

```
struct WFMacStatsStruct {
    uint32_t MibWEPExcludeCtr;
    uint32_t MibTxBytesCtr;
    uint32_t MibTxMulticastCtr;
    uint32_t MibTxFailedCtr;
    uint32_t MibTxRtryCtr;
    uint32_t MibTxMultRtryCtr;
    uint32_t MibTxSuccessCtr;
    uint32_t MibRxDupCtr;
    uint32_t MibRxCtsSuccCtr;
    uint32_t MibRxCtsFailCtr;
    uint32_t MibRxAckFailCtr;
    uint32_t MibRxBytesCtr;
    uint32_t MibRxFragCtr;
    uint32_t MibRxMultCtr;
    uint32_t MibRxFCSErrCtr;
    uint32_t MibRxWEPUndecryptCtr;
    uint32_t MibRxFragAgedCtr;
    uint32_t MibRxMICFailureCtr;
};
```

Members

Members	Description
uint32_t MibWEPExcludeCtr;	Number of frames received with the Protected Frame subfield of the Frame Control field set to zero and the value of dot11ExcludeUnencrypted causes that frame to be discarded.
uint32_t MibTxBytesCtr;	Total number of Tx bytes that have been transmitted
uint32_t MibTxMulticastCtr;	Number of frames successfully transmitted that had the multicast bit set in the destination MAC address.
uint32_t MibTxFailedCtr;	Number of Tx frames that failed due to the number of transmits exceeding the retry count.
uint32_t MibTxRtryCtr;	Number of times a transmitted frame needed to be retried
uint32_t MibTxMultRtryCtr;	Number of times a frame was successfully transmitted after more than one retransmission.
uint32_t MibTxSuccessCtr;	Number of Tx frames successfully transmitted.
uint32_t MibRxDupCtr;	Number of frames received where the Sequence Control field indicates a duplicate.
uint32_t MibRxCtsSuccCtr;	Number of CTS frames received in response to an RTS frame.
uint32_t MibRxCtsFailCtr;	Number of times an RTS frame was not received in response to a CTS frame.
uint32_t MibRxAckFailCtr;	Number of times an Ack was not received in response to a Tx frame.
uint32_t MibRxBytesCtr;	Total number of Rx bytes received.
uint32_t MibRxFragCtr;	Number of successful received frames (management or data)
uint32_t MibRxMultCtr;	Number of frames received with the multicast bit set in the destination MAC address.
uint32_t MibRxFCSErrCtr;	Number of frames received with an invalid Frame Checksum (FCS).
uint32_t MibRxWEPUndecryptCtr;	Number of frames received where the Protected Frame subfield of the Frame Control Field is set to one and the WEPOn value for the key mapped to the transmitter's MAC address indicates the frame should not have been encrypted.

uint32_t MibRxFragAgedCtr;	Number of times that fragments ‘aged out’, or were not received in the allowable time.
uint32_t MibRxMICFailureCtr;	Number of MIC failures that have occurred.

Description

Used in WF_GetMacStats.

1.9.2.10.1.6 WFMulticastConfigStruct Structure

File

drv_wifi_api.h

Syntax

```
struct WFMulticastConfigStruct {
    uint8_t filterId;
    uint8_t action;
    uint8_t macBytes[6];
    uint8_t macBitMask;
};
```

Description

Used in WF_MulticastSetConfig, WF_MulticastGetConfig.

1.9.2.10.1.7 tWFDeviceInfoStruct Structure

File

drv_wifi_api.h

Syntax

```
struct tWFDeviceInfoStruct {
    uint8_t deviceType;
    uint8_t romVersion;
    uint8_t patchVersion;
};
```

Members

Members	Description
uint8_t deviceType;	MRF24WB0M_DEVICE or MRF24WG0M_DEVICE
uint8_t romVersion;	ROM version number
uint8_t patchVersion;	Patch version number

Description

used in WF_GetDeviceInfo

1.9.2.11 Access Point Compatibility

Introduction * The MRF24WB0M / MRF24WG0M has passed through Wi-Fi.org certification testing. Not all routers pass through Wi-Fi.org certification, and some are pre-configured in Greenfield modes. Further, users can set configurations that severely limit performance or prevent communications. This section is intended to provide an on-going compatibility snapshot among a few of the most popular and market leading access points as well as a larger group of worldwide units. The test results will show the usability of the Microchip Wi-Fi modules operating with the latest release of the Microchip TCPIP stack.

Wi-Fi Alliance Testing

To carry the Wi-Fi Alliance logo, Wi-Fi products must successfully pass numerous tests, including compatibility testing. Wi-Fi

compatibility testing is performed against 4 representative access points, with a subset of tests run against each of the access points. Devices are tested against these access points for characteristics such as connectivity, security, throughput, and a breadth of other specifications. Microchip Wi-Fi modules have successfully passed the Wi-Fi Alliance testing. The report is titled WFA7150 and is available at http://certifications.wi-fi.org/pdf_certificate.php?cid=WFA7150

Additional Wi-Fi Compatibility Testing

Wi-Fi technology is dramatically expanding the reach and applications of the internet to embedded devices. In many cases, Wi-Fi is new to the markets and applications it is reaching. As a result, Microchip feels it is important to raise the bar on compatibility testing, and education of the developer.

Microchip has thus adopted the Wi-Fi.org test bench for more generic Access Point testing. The goal of these tests is to ensure basic connectivity in multiple non-secure and secure scenarios with a global representation of top selling access points.

Pass Criteria

The following tests are part of the current testing suite and must pass for the Access Point to be considered compatible.

- Following in conditions of no security, WEP40 and WEP104, WPA-PSK (TKIP), WPA2-PSK (AES)
- AP association, Iperf UDP upload/download, Iperf TCP upload/download, DHCP, ICMP ping

In many cases there are other modes that can be run with the Access Points and the user must take caution that if the mode is not listed, then compatibility is not necessarily guaranteed. These modes are usually Greenfield use, modes being deprecated by Wi-Fi.org, or cases of limiting the use of the Access Point for more private networking purposes and not for true Wi-Fi compatibility.

Examples of special modes not necessarily part of the results:

- WPA-PSK(AES) security: WPA-PSK security is defined as using TKIP. This is a mixed mode. This mode works if the AP just auto-detects and does not mix.
- WPA2-PSK (TKIP) security: WPA2-PSK security is defined as using 802.11i with AES. This is a mixed mode. This mode works if the AP just auto-detects and does not mix.
- 802.11g only, 802.11n only, 802.11g/n only: these are private network modes (cutting out mandatory support for 802.11b). These modes may work if basic rates are limited to 1&2mbps per 802.11.

List of compatible Access Points:

- 2Wire 1701HG
- 2Wire 2701HG-B
- 3COM 3CRWER100-75
- 3COM WL-524
- Actiontec GT704-WG
- Apple Airport Express
- Apple Airport Extreme
- Apple Time Capsule
- Asus RT-N16
- Asus WL530g
- AirLink AR690W
- Belkin N1

- Belkin F5D7231-4
- Belkin F5D8231-4
- Belkin F7D1301 v1
- Belkin F7D3302 v1
- Belkin F7D5301 v1
- Belkin Surf N300
- Buffalo WHR-G125
- Buffalo WHR-HP-G54
- Buffalo WHR-HP-GN
- Cisco E1000
- Cisco E3000
- Cisco E4200
- Cisco M20
- Cisco Vallet M10
- Corega CG-WLAPGMN
- Corega CG-WLBARGO
- D-Link DI-524
- D-Link DIR-615
- D-Link DIR-655
- D-Link DIR-665
- D-Link DIR-825
- D-Link DIR-855
- D-Link WBR-1310
- D-Link WBR-2310
- Dynex DX-WGRTR G
- Dynex DX-WGRTR v1000
- Level1 WBR-3408
- Linksys WRT150N v1.1
- Linksys WRT310N
- Linksys WRT54G2
- Microsoft MN-700
- Netgear WG103
- Netgear WGR614v9
- Netgear WGT624v2
- Netgear WN2000RPT
- Netgear WN802T v2
- Netgear WNDR3300
- Netgear WNDR3700
- Netgear WNR1000 v2
- Netgear WNR1000 v3
- Netgear WNR200 v3
- Netgear WPN824v2

- Netgear WNR854T
- PCI MZK-W04NU
- Proxim AP-700
- SMC Networks SMCWBR14S-N4
- SMC Networks SMCWBR14T-G
- TP-Link TL-WR340G
- TP-Link TL-WR541G
- TP-Link TL-WR740N
- TP-Link TL-WR741ND
- TP-Link TL-WR841ND
- TP-Link TL-WR941N/D
- Westell B90-327W15-06
- ZyXel P-330W
- ZyXel X550N

*Note Tests Performed:

- Basic association with the AP (no security)
- Association with WEP security
- Association with WPA/WPA2-PSK security
- Ping test validation.

1.9.2.12 802.11 AP/Router Configuration Settings

Describes some basic tips for setting up and configuring a WiFi network.

Description

Tips for Setting up Routers for 802.11b/g Use

The purpose of this section is to describe the settings for the most typical AP configurable parameters to enable compatibility with the Microchip MRF24WB0M / MRF24WG0M devices :

1. **DHCP Settings** - For DHCP on LAN side (where AP is DHCP server), set Router to Enable DHCP server. Set Client Lease time to be longer than the typical off time of the station to ensure that the IP address provided doesn't change each time the station is powered up. If an option for Always Broadcast is present for DHCP setup (broadcasts all DHCP responses to all clients), it should be disabled.
2. **Data Rate Settings** - Ensure that service rates include 802.11b, 802.11g or 802.11n only rates (green field) should be avoided, but mixed settings are usually acceptable. If a Basic Rate setting is defined, it should be set to 1 and 2MBPS only.
3. **SSID Broadcast** - Should typically be enabled so that the AP sends beacon frames containing the SSID. If disabled, ensure that Microchip Stack is set for Active Scanning.
4. **Channel Selection** - For debug purposes, it is typical to use a fixed channel instead of Auto Channel Selection. If a fixed channel has been selected for the MRF24 Station, select the corresponding channel for the AP.
5. **Multicast Passthrough** - If using multicast features (ZeroConfig for instance) ensure that the Router is configured to enable forwarding of Multicast packets.
6. **Beacon Interval** - Set the value for the time interval between AP beacons, typical is 100msec. For lower power, this can be set to a smaller value, say 30mS, if the DTIM interval is correspondingly increased.
7. **RTS Threshold** - Set the value for the frame size above which RTS/CTS will be used, typical is 2347.
8. **Fragmentation Threshold** - Set the value for the frame size above which packets will be fragmented, typical is 2346.
9. **DTIM Interval** - Set the value for Delivery Traffic Indication Message Interval, typical is 3 if the Beacon Interval is set for

100mSec. For lower power with the MRF24WB0M / MRF24WG0M, if the Beacon Interval is set to 30mS, then the DTIM should be set to 100 to allow 300mS DTIM Interval.

10. **WLAN Partition (or AP Isolation)**- Prevents AP clients from communicating to each other, typically disabled.
11. **WMM Enable** - Allows wireless multimedia traffic, disable unless necessary for other AP services.
12. **Short Guard Interval (GI)** - Lowers the guard interval between frames, disable unless necessary for other AP services.
13. **WiFi Protected Setup (WPS)** - Enables WPS device discovery, disable unless necessary for other AP services.
14. **Frame Burst** - Enables higher wireless packet throughput, disable unless necessary for other AP services. This may be called turbo, or other marketing terms.
15. **CTS Protection Mode** – Improves reliability of 802.11g traffic, disable unless necessary for other AP services.
16. **Key Entry** – Security can be entered with either a numerical key or an ASCII passphrase. Ensure you enter what the AP expects. If just starting, it is best to have another station like a laptop to validate what the AP is expecting.

1.9.2.13 WiFi Troubleshooting Tips

Lists some answers to some common Wi-Fi questions.

Description

The following clarifications are to be noted for use of the MRF24W with Microchip TCP/IP Stack versions unless otherwise noted.

For topics not addressed in this documentation, search [Microchip Technical Support Knowledgebase](#).

1.9.2.13.1 Null String ESSID

Describes how to handle null string ESSID

Description

Null String ESSID

It is possible to call `WF_CMConnect(cpld)` with a `cpld` of zero. If this happens, the connection manager can use erroneous values for the SSID, Network Mode, Security configuration, etc. which will cause the module to connect to a wrong AP or not connect at all. The only valid values that can be used for connection profile references are 1 and 2 (assuming that the `WF_CPCreate(&cpld)` succeeded in creating these profile references prior to the attempted connection).

Work around:

When creating a connection profile, verify that the profile number returned is always either 1 or 2. If the returned value is 0, delete the profile and recreate it. When connecting with `WF_CMConnect(cpld)`, ensure that only a valid profile number previously returned from `WF_CPCreate(&cpld)` is used.

1.9.2.13.2 Read back RF module Firmware version

Describes how to read back RF module firmware version

Description

RF module FW version

There are 2 methods available.

1. Run WiFi Console standalone CLI command

Type in command `getwfvver`.

The following will be displayed; MRF24W firmware version and Host Driver version.

2. Invoking function prototype `WF_GetDeviceInfo(tWFDeviceInfo *p_deviceInfo)`

As part of initialization, `WF_Init()` will call `WF_GetDeviceInfo(tWFDeviceInfo *p_deviceInfo)`.

3 parameters will be returned

- Type (1 for MRF24WB0M_DEVICE and 2 for MRF24WG0M_DEVICE)
- Rom Version (0x12 for MRF24WB0M and 0x30 / 0x31 for MRF24WG0M)
- Patch version

MRF24WG0M RF module FW version

From RF module FW version 0x3107 onwards, FW release will follow this roll-out order

Even Numbered (eg 0x3108, 0x310a, etc)

- Multi-DHCP
- WPA-EAP
- No Wi-Fi Direct
- All other features, including SoftAP supporting max 4 clients

Odd Numbered (eg 3109, 0x310b)

- No multi-DHCP
- No WPA-EAP
- Wi-Fi Direct
- All other features, including SoftAP supporting only 1 client

1.9.2.13.3 RF Module Firmware Update

Describes how to update MRF24WB0M/MRF24WG0M RF module firmware

Description

Flash Update Project

This is applicable for both MRF24WB0M and MRF24WG0M.

Go to website

<ftp://mrfupdates@ftp.microchip.com>

where username is mrfupdates and password is mchp1234.

For MRF24WB0M, the flash update project is MRF24WB_Exp16FlashUpdater-120c-Rev1-windows-installer.exe

For MRF24WG0M, the flash update project is MRF24WG_FlashUpdater-3107-Rev1-windows-installer.exe

Over-The-Air (OTA) MRF24WG0M RF Module Firmware Update

This is applicable only for MRF24WG0M.

To use OTA RF module firmware update, MRF24WG0M needs at least RF module FW version 0x3107 and MLA v5.42.04 Oct 2012 release or later.

This method is located in MPLABX tcpip_wifi_console project.

In tcpip.h, enable STACK_USE_AUTOUPDATE_TCPCLIENT.

In drv_auto_update_tcp_client_24g.c, configuration is defined in this file.

```
static BYTE ServerName[] = "www.microchip.com";
static BYTE PatchName[]="/mrfupdates/A2Patch_3107.bin";
//Username is mrfupdates , password is mchp1234
static BYTE Key_authorization[]="bXJmdXBkYXRlc3ptY2hwMTIzNA==" ;
static WORD ServerPort = 80; // Defines the port to be accessed for this application
```

Update PatchName[] with the required MRF24W firmware version file name. PatchName is case-sensitive.

In drv_wifi_config.h, configure Wi-Fi parameters according to selected AP/router.

- Define infrastructure network type

```
#define MY_DEFAULT_NETWORK_TYPE CFG_WF_INFRASTRUCTURE
```

- Define SSID and Wi-Fi security mode as that used by the AP/router.

```
#define MY_DEFAULT_WIFI_SECURITY_MODE
```

```
#define MY_DEFAULT_SSID_NAME
```

- For faster connection, you may define the channel AP/router is in.

e.g. #define MY_DEFAULT_CHANNEL_LIST {3}

1.9.2.13.4 Wi-Fi Protected Setup (WPS) Issues

Describes MRF24WG0M WPS Issues

Description

Wi-Fi Protected Setup (WPS) security connection mode is only supported by MLA v5.42 July 2012 releases or later.

- **Why is MRF24WG0M reporting error message "Event: Connection Failed : WF_RECV_DISASSOC : WF_UNSPECIFIED" when trying to connect to certain AP/routers in WPS-PBC security connection modes ?**

When running WiFi TCP/IP Demo, the output display shows

```
*** WiFi TCP/IP Demo ***
```

```
Start WiFi Connect
```

```
Domain: FCC
```

```
MAC: 00 1E C0 08 F1 04
```

```
SSID: (none)
```

```
Network Type: Infrastructure (using WPS Push Button)
```

```
Scan Type: Active Scan
```

```
Channel List: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
```

```
Retry Count: Retry Forever
```

Beacon Timeout: 40

Security: WPS push button method

Power Save: Disabled

New IP Address: 169.254.1.1

Event: Connection Failed : WF_RECV_DISASSOC : WF_UNSPECIFIED

A wireless capture revealed that some AP/routers is transmitting disassociation frames instead of deauthentication frames during WPS process. This behavior is not according to 802.11 specifications and thereby causing WPS to dysfunction. RF module FW (from 0x3108 onwards) was modified to handle this erroneous case.

- **Why is the AP/router, in WPS mode, taking a longer time to connect, as compared to when AP/router is in WPA-PSK/WPA2-PSK security mode?**

Refer to Wi-Fi Protected Setup Specification Version 1.0h standard.

WPS protocol is built on top of the WPA/WPA2 EAPOL 4-way handshaking process and there are additional process for WPS. Time allowed for entire WPS protocol is 2 min.

The specifications listed the below parameters

1. Retransmission timeout = 5 sec
2. Individual message processing timeout = 15 sec
3. Overall timeout for entire protocol to complete = 2 min

- **What is needed to see the WPS frame exchanges in a wireless sniffer capture?**

Either click on <Protocol> field and scroll down until this starts with the alphabet "E..." or in the <Filter:> field, type in "eapol".

- **What is special about the PIN selected for WPS-PIN? Can this PIN be randomly selected by the user?**

The 8 digit PIN is NOT randomly generated. The last digit is the checksum of first 7 digits of the PIN. If this checksum is wrong, MRF24WG0M module will reject this PIN code.

- **I just bought a brand new AP and have problems using WPS to connect to MRF24WG0M.**

It has come to our attention that there are some AP/routers that do not work out of the box using WPS feature. These AP/routers may still require some minimum set up, such as setting up security to WPA2-Personal, etc. Please refer to their instruction manuals for more details in WPS setup.

1.9.2.13.5 Network Switch or Change

Describes how to perform a network switch or change

Description

Network Switch or Change

WiFi EasyConfig is a good demo reference for handling network switch.

In WiFi EasyConfig, the function prototype `WFEasyConfigProcess()` is called to execute a network change.

MRF24W is put into Hibernate mode, which implies a reset operation, whereby the LDO is turned off.

Old connection profile is deleted and new connection profile is created. New parameters, linked to selected network, are configured into new profile. Then exit out of Hibernate mode.

The recommend sequences are

1. Disconnect from current network by invoking `WF_CMDDisconnect()`
2. Delete the profile by invoking `WF_CPDelete()`
3. Create new profile by invoking `WF_CPCreate()`
4. Set up parameters for this new profile such as setting SSID, Wi-Fi security, network type.
5. Enter Hibernate mode by setting `WF_hibernate.state = WF_HB_ENTER_SLEEP` and `WF_hibernate.wakeup_notice = FALSE`
6. Have a short time delay such as `DelayMs(50)`
7. Exit from Hibernate mode by setting `WF_hibernate.wakeup_notice = TRUE`

1.9.2.13.6 Hibernate Mode

Describes the MRF24W RF module Hibernate Mode software flow

Description

Hibernate Mode

Hibernate mode is used during network switch or change.

Ensure definition `WF_USE_POWER_SAVE_FUNCTIONS` is enabled.

Within the `main()` loop, `StackTask()` will call `MACProcess()`. It will then invoke `CheckHibernate()`, which executes/handles hibernate mode based on `WF_hibernate.state` and `WF_hibernate.wakeup_notice`.

To enter into Hibernate mode, the following settings are required

- `WF_hibernate.state = WF_HB_ENTER_SLEEP`
- `WF_hibernate.wakeup_notice = FALSE`

To exit from Hibernate mode, the following settings are required

- `WF_hibernate.wakeup_notice = TRUE`

1.9.2.13.7 Management Scan Message Conflict

Describes how to handle management scan message conflict

Description

Management scan message conflict

Management messages must always return successful or it causes an assert in the host driver. An unsuccessful management message can occur when the connection retry is enabled (`MY_DEFAULT_LIST_RETRY_COUNT > 0`) causing the device to be scanning due to a dropped connection, and then a disconnect, or connect, or scan command is sent.

Work around:

If you are controlling connect/reconnect from the host actively, then disable all firmware retry by using “no scan retry” and “no de-authorization action”.

a. To disable Scan Retry

```
WF_CASetListRetryCount(MY_DEFAULT_LIST_RETRY_COUNT); should be 0
```

b. To disable De-authorization action

```
WF_CASetDeauthAction(WF_DO_NOT_ATTEMPT_TO_RECONNECT);
```

c. To disable De-authentication action

```
WF_CASetBeaconTimeoutAction(WF_DO_NOT_ATTEMPT_TO_RECONNECT);
```

d. Use “Connect” only on “permanent loss” or “connection failure”.

e. To do a Scan, first check the firmware state first by using WF_CMGetConnectionState()

i. If the return state is WF_CSTATE_NOT_CONNECTED (or WF_CSTATE_CONNECTION_PERMANENTLY_LOST), then this means firmware is in IDLE, so host can issue host scan safely

ii. If the return state is WF_CSTATE_CONNECTED_INFRASTRUCTURE, then this means firmware is in CONNECTED. In this case a scan command can be issued but a watchdog timer must be used to time for conflict. Also, ensure the management timer is set for at least 0.4seconds per channel scanned to prevent queued Tx buffer requests from timing out.

iii. If return state is WF_CSTATE_CONNECTION_IN_PROGRESS (or WF_CSTATE_RECONNECTION_IN_PROGRESS), then this means firmware is in the middle of connection process and a scan must not be initiated.

f. If “Disconnect” function is desired, a watchdog timer needs to be used to address the case where a conflict occurs with an over the air disassociate or deauthorize.

g. For watchdog timing, advised timing is 2x the management packet timeout (that is, use 4seconds unless the management timeout has been increased).

b. If you are only using the firmware retry and not doing ANY connection management (scan, connect, idle, etc.) then you can use MY_DEFAULT_LIST_RETRY_COUNT>0 or retry forever (MY_DEFAULT_LIST_RETRY_COUNT=255). If you lose connection, you can reconnect using the “connect” API. Do not use “Disconnect”.

a. If “Disconnect” function is desired, a watchdog timer needs to be used to address the case where a conflict occurs with an over the air disassociate or deauthorize.

1.9.2.13.8 Handling of maximum length SSID

Describes how to handle maximum length SSID

Description

Maximum length SSID

An issue has been found with MLA v5.42.06 Feb 2013 and prior releases, when SSID of the selected AP is up to the maximum length (length of 32). If SSID of AP is 32 characters, then the type of security is always seen as OPEN. However, if SSID is less than 32 characters, then host-scan result returns the correct security type.

The root cause has been found to occur during the process of converting the SSID to ASCII string in order to print this out through console, whereby the last character was set to 0 to indicate end of string.

The workarounds needed are as follows.

1. drv_wifi_easy_config.c

In WFRetrieveScanResult(), update the code as

```
UINT16 WFRetrieveScanResult(UINT8 Idx, tWFScanResult *p_ScanResult)
{
    if (Idx >= SCANCXT.numScanResults)
        return WF_ERROR_INVALID_PARAM;
    WF_ScanGetResult(Idx, p_ScanResult);
    return WF_SUCCESS;
}
```

In WFDisplayScanMgr(), update the code as

```
void WFDisplayScanMgr()
{
    tWFScanResult bssDesc;
    char ssid[WF_MAX_SSID_LENGTH+1];
    char rssiChan[48];
    int i;
    .....
    /* Display SSID */
    for(i=0;i<WF_MAX_SSID_LENGTH;i++) ssid[i] = bssDesc.ssid[i];
    ssid[WF_MAX_SSID_LENGTH] = 0;
    putsUART(ssid);
    putsUART("\r\n");
    .....
}
```

2. custom_http_app.c

Modify HTTPPrint_name() as follows.

```
void HTTPPrint_name(void)
{
    if (bssDescIsValid)
    {
        if(strlen((const char*)bssDesc.ssid)<WF_MAX_SSID_LENGTH)
            TCPPutString(sktHTTP, bssDesc.ssid);
        else
        {
            unsigned char buf_tmp[WF_MAX_SSID_LENGTH + 1];
```



```

int i;
for(i=0;i<WF_MAX_SSID_LENGTH;i++) buf_tmp[i] = bssDesc.ssid[i];
buf_tmp[WF_MAX_SSID_LENGTH] = 0;
TCPPutString(sktHTTP, buf_tmp);
}
}
else
{
TCPPutROMString(sktHTTP, (ROM BYTE *)"0"); }

```

1.9.2.13.9 Multicast Filters : Hardware vs Software

Describes considerations for hardware and software Multicast filters.

Description

Multicast Filters

MRF24W has 2 hardware multicast filters.

If your design requires more than 2 multicast filters, there is an option to extend the multicast filters to a maximum of 16 software-based multicast filters.

To use this maximum of 16 multicast filters, the macro

```
#define ENABLE_SOFTWARE_MULTICAST_FILTER
```

needs to be enabled. The MRF24W FW will be notified to use software to run the filters instead of hardware. The downside of this software-based multicast filter option is the performance could possibly be degraded in the case scenario when there are so many multicast packets on the air.

1.9.2.13.10 MRF24WB0M assert failures whe using <iwconfig scan> command

In WiFi console demo, using MRF24WB0M and certain sequences (wrong pass-phrase), <iwconfig scan> command can produce assert failures.

Description

Host scan <iwconfig scan> command ASSERT failures

Applicable for MRF24WB0M only.

Using both MLA v2012-04-03 and v2013-02-15, if all connection parameters are correct, with only the pass-phrase set wrongly, it causes assert failure

WF ASSERTION at drv_wifi_mgmt_msg.c Line Number = 248.

However, if the wrong ssid is used, <iwconfig scan> is OK.

- Case [1] wrong ssid only. < iwconfig scan > is OK
- > iwconfig ssid MCHP_test
- > iwpriv enc wpa-phrase
- > iwpriv phrase 12345434

```
> iwconfig mode managed
> Event: Connection Failed – eventInfo = 10, WF_NO_SUI_TABLE_AP_FOUND_FAILURE
> iwconfig scan <---- iwconfig scan is OK
Scanning...
Scan completed.
> Event: Scan Results Ready, 21results
```

- Case [2] wrong pass-phrase only. < iwconfig scan > causes assert failure

```
> iwconfig ssid MCHP_test
> iwpriv enc wpa-phrase
> iwpriv phrase 12341234
> iwconfig mode managed
> Event: Connection Failed – eventInfo = 9, WF_SECURITY_MISMATCH_FAILURE
> iwconfig scan
Scanning...
WF ASSERTION at drv_wifi_mgmt_msg.c Line Number = 248
```

Recommended workarounds

1. drv_wifi_scan.c

Replace WF_Scan() with the following

```
UINT16 WF_Scan(UINT8 Cpld)
{
    UINT8 hdr[4];
    #ifndef MRF24WG
    UINT8 connectionState;
    UINT8 dummy;
    #endif

    if (!WF_CMIsHostScanAllowed())
        return WF_ERROR_OPERATION_CANCELLED;

    #ifndef MRF24WG
    WF_CMGetConnectionState(&connectionState, &dummy);
    if (connectionState == WF_CSTATE_NOT_CONNECTED)
        WF_CMConnect(0xff); /* MRF24WB 0x120c host scan bug workaround */
    #endif
```

```

hdr[0] = WF_MGMT_REQUEST_TYPE;
hdr[1] = WF_SCAN_START_SUBTYPE;
hdr[2] = CpId; /* Connection Profile ID */
hdr[3] = 0; /* not used */

SendMgmtMsg(hdr, /* header */
sizeof(hdr), /* size of header */
NULL, /* no data */
0); /* no data */

/* wait for mgmt response, free it after it comes in (no data needed) */
WaitForMgmtResponse(WF_SCAN_START_SUBTYPE, FREE_MGMT_BUFFER);

return WF_SUCCESS;
}

```

2. drv_wifi_mgmt_msg.c

In WaitForMgmtResponse()

replace

} else {

WF_ASSERT(hdr.result == WF_SUCCESS);

}

with

} else {

if (!(hdr.result == WF_ERROR_CP_INVALID_PROFILE_ID

&& hdr.subtype == WF_CM_CONNECT_SUBTYPE)) {

WF_ASSERT(hdr.result == WF_SUCCESS);

1.9.2.13.11 MRF24WB0M advertised supported rates of 1, 2, 5.5 and 11 Mbps

Describes why MRF24WB0M shows advertised supported rates of 1, 2, 5.5 and 11 Mbps

Description

MRF24WB0M, on the product level, only supports legacy 802.11 data rates of 1Mbps and 2Mbps.

However when MRF24WB0M transmits an association request to an AP, MRF24WB0M will advertise in the supported rates information element (IE) rates of 1, 2, 5.5 and 11 Mbps.

This is not a bug and is intentional, leveraging from our past experiences.

If MRF24WB0M module just advertises 1,2 Mbps, most APs out in the field have a tendency to drop MRF24WB0M module and not permit connection to the MRF24WB0M module. Therefore for APs compatibility reasons, MRF24WB0M will advertise supported rates reflecting that of the APs. For example, the beacon from the AP will advertise supported rates of 1,

2, 5.5, 11, 18, 24, 36 and 54 Mbps. MRF24WB0M will duplicate the 802.11b rates in this information and advertise supported rates of 1, 2, 5.5 and 11 Mbps in the association request frame supported rates IE field.

However, the tradeoff is that APs, thinking that MRF24WB0M module supports 1,2,5.5, 11 Mbps, starts transmitting 11 Mbps and this will cause eventually wireless connectivity to fail as MRF24WB0M module will not be able to receive the AP's transmissions. However, bear in mind, according to 802.11 specifications, broadcast frames are to be transmitted at base rates of 1 and 2 Mbps. At the same time, this trend may start changing as the push for higher rates persist.

Bear in mind that MRF24WB0M module is meant for a captive environment, whereby the 802.11 network is known.

1.9.2.13.12 MRF24WB0M Compatibility with AP/Routers

Describes MRF24WB0M compatibility with AP/Routers

Description

MRF24WB0M Compatibility With AP/Routers

MRF24WB0M supports only legacy 802.11b rates of 1Mbps and 2Mbps.

To ensure compatibility with MRF24WB0M,

- iOS and Android FW releases need to support 1 and 2 Mbps
- APs/ routers need to support 1 and 2 Mbps

Recommended configurations on APs/Routers settings

- Set AP/router basic rate to 1-2 Mbps
- Set AP/router channel settings to defined social channels such as channel 1, 6 or 11.

If iOS, Android FW releases and AP/routers do not support 1 and 2 Mbps, customers are encouraged to feedback to the manufacturers or developers.

MRF24WB0M is best suited for captive networks, whereby the surrounding wireless networks are under control by the users.

In applications where the surrounding wireless networks are unknown and constantly changing, the MRF24WG0M is highly recommended over the MRF24WB0M.

1.9.2.13.13 Encounter issues after upgrading MRF24WB0M RF module Firmware version 0x1207

Describes new "gRFModuleVer1209orLater" APIs

Description

Current MRF24WB0M RF module Firmware version is 0x120C.

Starting with MRF24WB0M RF module Firmware version 0x1209, a new compilation parameter "gRFModuleVer1209orLater" is introduced into MLA TCPIP source codes. Ensure these supporting source codes are incorporated.

"gRFModuleVer1209orLater" changes are

1. New APIs:

These new APIs are not backward compatible with MRF24WB0M RF module Firmware version 0x1207.

- `WF_CPSetWepKeyType()` // Shared or Open Key
- `WF_CMGetConnectContext()` // Retrieve AP channel & bssid
- `WFEnableBroadcastProbeResponse()` // Send Probe Response with broadcast address in destination address.
- `WFEnableAggressivePowerSave()` // Turn off RF quicker in PS mode.
- `WF_CPSetSsidType()` // Allows connection to hidden SSID in Adhoc network.
- `WFEnableDeferredPowerSave()` // To enable compatibility with 1207. Allows FW to wait till DHCP done before going to PS mode.
- `WF_FixTxRateWithMaxPower()` // Set TX rate at 1Mbps with Max power

1.9.2.13.14 How to fix MRF24WB0M / MRF24WG0M transmission rates

Describes how to fix the transmission rates of MRF24W

Description

MRF24W Transmission Rates

The function prototype `WF_FixTxRateWithMaxPower()` can be used to fix the transmission rates, with maximum power, for the MRF24W.

Even though in the MLA SW this function is compiled only under MRF24WB configuration, this function is applicable also for MRF24WG0M.

For MRF24WB0M, only 1 or 2 Mbps transmission rates can be fixed.

For MRF24WG0M, this function can be used to fix the transmission rates beyond 1 or 2 Mbps. The input parameter to be used is

```
* #define kOneMbps (2)
* #define kTwoMbps (4)
* #define kFiveMbps (11)
* #define kElevenMbps (22)
* #define kSixMbps (12)
* #define kNineMbps (18)
* #define kTwelveMbps (24)
* #define kEighteenMbps (36)
* #define kTwentyFourMbps (48)
* #define kThirtySixMbps (72)
* #define kFortyEightMbps (96)
* #define kFiftyFourMbps (108)
```

Bear in mind, this fixed transmission rate only applies to data packets. All management packets still use 1 Mbps.

1.9.2.13.15 How to determine new IP address assigned

Describes how to determine new IP address being assigned to MRF24W by a DHCP server

Description

Determine New IP Address Assigned to MRF24W

Check the global variable `g_DhcpSuccessful`.

When `g_DhcpSuccessful` becomes TRUE, `AppConfig.MyIPAddr` will reflect the new IP address that is assigned to MRF24W.

Below are some suggestions to determine the new IP address that has been assigned.

a) Enable Zero Configuration / mDNS

This feature enables users not to have any knowledge of the IP Address. Instead what is needed is the URL address such as `http://mchpboard.local`.

After typing this URL address, the MRF24W web page could be displayed, showing the new IP address.

Example

b) AP DHCP Client Table

Refer to your AP's documentation for the necessary steps required to obtain the new IP address. As an example, for certain APs, by selecting <Status> and then <Local Network>, an option <DHCP Client Table> is offered. This DHCP Client Table will list the IP address that is assigned to the MRF24W.

c) Wi-Fi TCP/IP Demo will display the new IP address through a designated console terminal.

1.9.2.13.16 How to increase TCP throughput

Describes the various methods to increase TCP throughput

Description

Methods to increase TCP throughput

1. Enlarge generic TCP rx/tx buffer size. Refer to `tcpip_config.h`.

In `Tcpip_config.h` modify the buffer size accordingly.

For example,

```
{TCP_PURPOSE_GENERIC_TCP_CLIENT, TCP_ETH_RAM, 1024, 100},
{TCP_PURPOSE_GENERIC_TCP_SERVER, TCP_ETH_RAM, 20, 4096},
```

2. In `tcp.c`, increase max TCP segment size.

For example, change `#define TCP_MAX_SEG_SIZE_RX` from (536u) to (1460u).

3. Disable TCP rx checksum check. This is not necessary because packet integrity is guaranteed by the MAC layer CRC32 check.

```
BOOL TCPPProcess(NODE_INFO* remote, IP_ADDR* localIP, WORD len) {
```

```
TCP_HEADER TCPHeader;
```

```
PSEUDO_HEADER pseudoHeader;
```

```
WORD_VAL checksum1;
```

```
WORD_VAL checksum2;
```

```
BYTE optionsSize;
```

```
#if 0 // disable TCP RX checksum check
```

```
// Calculate IP pseudoheader checksum.
```

```

pseudoHeader.SourceAddress = remote->IPAddr;
pseudoHeader.DestAddress = *localIP;
pseudoHeader.Zero = 0x0;
pseudoHeader.Protocol = IP_PROT_TCP;
pseudoHeader.Length = len;
SwapPseudoHeader(pseudoHeader);
checksum1.Val = ~CalcIPChecksum((BYTE*)&pseudoHeader,
sizeof(pseudoHeader));
// Now calculate TCP packet checksum in NIC RAM - should match
// pseudo header checksum
checksum2.Val = CalcIPBufferChecksum(len);
// Compare checksums.
if(checksum1.Val != checksum2.Val)
{
MACDiscardRx();
return TRUE;
}
#endif

```

4. Use PIC ram as TCB buffer instead of MRF24W scratch memory.

In tcpip_config.h

for example

Change

```
{TCP_PURPOSE_FTP_DATA, TCP_ETH_RAM, 0, 128},
```

to

```
{TCP_PURPOSE_FTP_DATA, TCP_PIC_RAM, 0, 4096},
```

1.9.2.13.17 Missing DHCP Client Name

Describes the software fixes to display DHCP client name

Description

Display DHCP client name

When MRF24W is a client in the infrastructure network, MRF24W client name appears as empty field in the AP/router's DHCP client table . See figure below.

To have MRF24W display a DHCP client name, incorporate the following changes into dhcp.c

Add new code below.

```
#define DHCP_HOSTNAME_SIZE 18
```

```
char Dhcp_HostName[DHCP_HOSTNAME_SIZE + 1];
```

```
/******
```

Function:

```
static void DHCPSetHostName(char *HostName)
```

Description:

Set the Host Name.

Precondition:

None.

Parameters:

HostName = Pointer to zero terminated host name

Returns:

None

```
*****/
```

```
void DHCPSetHostName(char *HostName)
```

```
{
strncpy(Dhcp_HostName, HostName, DHCP_HOSTNAME_SIZE);
Dhcp_HostName[DHCP_HOSTNAME_SIZE] = 0;
}
```

Modify existing DHCPSEND() as below.

```
static void _DHCPSEND(BYTE messageType, BOOL bRenewing)
```

```
{
.....

// Load our interested parameters
// This is hardcoded list. If any new parameters are desired,
// new lines must be added here.
UDPPut(DHCP_PARAM_REQUEST_LIST);
UDPPut(DHCP_PARAM_REQUEST_LIST_LEN - 1);
UDPPut(DHCP_SUBNET_MASK);
UDPPut(DHCP_ROUTER);
UDPPut(DHCP_DNS);

// Add requested IP address to DHCP Request Message
if( (messageType == DHCP_REQUEST_MESSAGE) && !bRenewing) ||
((messageType == DHCP_DISCOVER_MESSAGE) && DHCPClient.tempIPAddress.Val))
{
UDPPut(DHCP_PARAM_REQUEST_IP_ADDRESS);
UDPPut(DHCP_PARAM_REQUEST_IP_ADDRESS_LEN);
```



```

UDPPutArray((BYTE*)&DHCPClient.tempIPAddress, DHCP_PARAM_REQUEST_IP_ADDRESS_LEN);
}

// Add any new parameter request here.
UDPPut(DHCP_HOST_NAME);

sprintf((char *)Dhcp_HostName,"%s_%02x%02x",    MY_DEFAULT_HOST_NAME,    AppConfig.MyMACAddr.v[4],
AppConfig.MyMACAddr.v[5]);

UDPPut(strlen(Dhcp_HostName));

UDPPutArray((BYTE*)Dhcp_HostName, strlen(Dhcp_HostName));

.....
}

```

With these modifications, MRF24W has a valid client name now. See figure below.

1.9.2.13.18 Error Scenario And Possible Causes

List of error messages and the possible causes

Description

- **During scanning, the web browser reported error message "Command failed, connection to development board was lost".**

This is not a bug because when you click the "OK" button in the message box, it will continue to work correctly. Above timeout error message happens when you choose to scan ALL channels. If you elect to scan only 2-3 channels, the scan duration will be much faster and the web browser is unlikely to timeout.

A possible solution is to increase the timeout setting in mchp.js

Eg \EasyConfigWebPages\javascript\mchp.js

```
var timeOutMS = 5000; //ms
```

- **Scan results seem to display 1 SSID less. For example, there are 10 scan results but only 9 scan results are displayed.**

If the scan result yields 10 scan results, our MRF24W + PIC development board will send 10 scan results to the webpage. However, the webpage will display only 9 scan results and drop its own SSID (scan result). As an example, the development board and laptop are using an AP named "test", then the webpage will not display SSID "test" among the displayed scan results.

- **Why is my device not displaying MRF24WG0M softAP SSID in the scan results?**

There are 2 possible reasons.

(a) Hidden SSID

(b) The device is only supporting active scan. MRF24WG0M RF module FW version 0x3107 only supports softAP with passive scan. From MRF24WG0M RF module FW version 0x3108 and future releases, softAP supports both active and passive scan. Check that your MRF24WG0M RF module FW version is 0x3108 and later.

- **How can the received signal strength indicator (RSSI) be obtained from the MRF24W?**

RSSI can only be obtained from the scan results `p_scanResult->rssi`. Refer to function prototype `WF_ScanGetResult()`. MRF24W checks out the signal strength from the preamble of the incoming packets. The higher the values, the stronger is the received signal strength.

MRF24WB : RSSI_MAX (200) , RSSI_MIN (106).

MRF24WG : RSSI_MAX (128) , RSSI_MIN (43).

The RSSI value is not directly translated to dbm because this is not calibrated number. However, as a guideline, MAX(200) corresponds to 0 dbm, MIN (106) corresponds to -94 dbm.

- **Why is MRF24W failing after chip reset?**

It is discovered that glitches on the MRF24W reset line. that is occurring after ChipReset(), could potentially cause MRF24W to fail.

The root cause is traced to the macros WF_SetCE_N and WF_SetRST_N, where the pin is configured as output first and then the level is set. The correct sequences should be to set the level first and then configure the pin as output.

```
#define WF_SetCE_N(level)
```

```
/* set pin to desired level */
```

```
WF_HIBERNATE_IO = level;
```

```
/* configure I/O as output */
```

```
WF_HIBERNATE_TRIS = 0
```

```
#define WF_SetRST_N(level)
```

```
/* set pin to desired level */
```

```
WF_RESET_IO = level;
```

```
/* configure the I/O as an output */
```

```
WF_RESET_TRIS = 0
```

The above changes apply to both MRF24WB0M and MRF24WG0M.

Refer to MLA v5.43.0 for the changes in the file drv_wifi_prev_24g.h (MRF24WG0M)

Refer to MLA v5.43.0 for the changes in the file drv_wifi_prev.h (MRF24WB0M)

- **Why is the software hanging in infinite loop within WaitForMgmtResponse() function ?**

For MLA v5.41 2012-02-15 and earlier versions, it is possible to be caught in an infinite loop. This could be caused by a race condition where a data message comes in before a mgmt response. Thus the workaround solution is to throw away a data message that comes in while waiting for a mgmt response.

This workaround is implemented in MLA v5.41.02 2012-04-03 and future versions.

The work-around is to change the while loop code to:

```
while (gMgmtConfirmMsgReceived == FALSE) {
```

```
WFProcess();
```

```
/* if received a data packet while waiting for mgmt packet */
```

```
if (g_HostRAWDataPacketReceived) {
```

```
// throw away the data rx
```

```
RawMountRxBuffer();
```

```
DeallocateDataRxBuffer();
```

```
g_HostRAWDataPacketReceived = FALSE;  
/* ensure interrupts enabled */  
WF_EintEnable(); }}
```

1.9.2.14 Wireless Packets Analysis

Describes the wireless packets transactions according to 802.11 connection protocols

Description

This section is applicable to any wireless 802.11 products and describes the client device in an infrastructure network type. The approach remains similar when analyzing other network types.

Hardware

Wireless sniffer hardware, such as AirPcap USB adaptor by Riverbed Technology, is needed to capture the wireless packets. Here it is assumed the user has knowledge of setting up the wireless sniffer hardware for packets capture.

Software

Software such as Wireshark is needed to analyse the wireless packets captured.

Even if you do not have the hardware and only have the wireshark capture file (with file extension *.pcapng), this software is still needed to view and analyze the wireless packets capture file.

To capture 802.11 wireless packets, a sniffer hardware such as the AirPcap is required. Certain configurations will not allow you to capture 802.11 wireless packets.

- A laptop/PC that has wireless network capabilities or wireless adaptor with an AP connected to this laptop/PC via wired Ethernet cable is not sufficient as this environment is likely to be in non-promiscuous mode. . This means that many packets will be filtered and not shown on the wireshark trace. If using wired ethernet or the built-in PC NIC, generally only packets meant for the PC are displayed. The AirPcap allows selecting a promiscuous NIC that will display all traffic in the air.
- A laptop/PC that has wireless network capabilities or wireless adaptor and only plugged into wired Ethernet. This will only allow monitoring of Ethernet traffic and specifically traffic that is targeted to the PC, even though this laptop/PC is 802.11 wireless connected to some wireless devices. Also this method will prevent seeing packets that are attempting to make it to the PC or anywhere else but are not getting through due to any misconfigurations.

Below figure shows the recommended configuration for wireless packets capture.

802.11 Protocols

Below figure shows the 802.11 exchange protocol for a client device in an infrastructure network.

Authentication and association frame types are unidirectional. For example, the client device will be the one transmitting the association request frame and likewise the AP/router will be the one to respond back with the association response frame.

Display Filter

Use the Wireshark display filter to filter down to the traces of interest. Refer to Wireshark website for more information on display filter settings.

As an example, enter into the display filter

```
(wlan.bssid == Cisco-Li_27:5a:a0 && wlan.da == Broadcast) || wlan.addr == Microchi_02:75:7e
```

In our example, Cisco-Li_27:5a:a0 represents the AP/router and Microchi_02:75:7e represents the wireless client.

- More examples of display filter

Wlan.fc.type == 0 Management frames

Wlan.fc.type == 1 Control frames

Wlan.fc.type == 2 Data frames

Wlan.fc.type_subtype == 0 Association request

Wlan.fc.type_subtype == 1 Association response

Wlan.fc.type_subtype == 2 Reassociation request

Wlan.fc.type_subtype == 3 Reassociation response

Wlan.fc.type_subtype == 4 Probe request

Wlan.fc.type_subtype == 5 Probe response

Wlan.fc.type_subtype == 8 Beacon

Wlan.fc.type_subtype == 10 Disassociation

Wlan.fc.type_subtype == 11 Authentication

Wlan.fc.type_subtype == 12 Deauthentication

Information you are likely to need for the display filter

- BSSID of AP/router

BSSID is unique and identifies a specific AP/router eg Cisco-Li_27:5a:a0

If you are unable to determine the BSSID, an alternative is to review the list of beacons in the wireless packets capture and use this list to narrow down the BSSID.

This will translate to "wlan.bssid == Cisco-Li_27:5a:a0" in the display filter.

- MAC Address of client device eg MRF24W

This will identify the client device's hardware address.

As an example, the console terminal when using the MRF24W will display a MAC address field.

See below figure where the MAC address is 00:1e:c0:02:75:7e.

"00:1e:c0:02:75:7e" is translated to "Microchi_02:75:7e".

Enter "wlan.addr == Microchi_02:75:7e" in the display filter.

- Source Address or Destination address

This could be the following; BSSID field, address field or "Broadcast" or "Multicast".

Beacons are broadcast frames and the destination address is set as "Broadcast".

As an example, to display beacons transmitted by a particular AP/router, this will translate to "(wlan.bssid == Cisco-Li_27:5a:a0 && wlan.da == Broadcast)" in the display filter.

Procedures To Analyze 802.11 wireless packets traces

Step 1 : Open the wireless packets capture traces / file.

Step 2: Use the display filter to filter down to the traces of interest.

Step 3 : Active or Passive Scanning

Before wireless connection is established, a client device needs to locate the specific AP/router that it wishes to join by either passive or active scanning.

Passive Scanning

Every “unhidden” AP/router will be transmitting beacons almost every beacon interval (BI). The client device is likely to receive multiple beacons from different AP/routers. A scan list result will be generated showing wireless devices or AP/routers that is present in the wireless network.

Each beacon frame will contain information about the infrastructure network such as

- o Transmission rate of beacon (Based on 802.11-2007 specifications, transmitted at 1-2Mbps)

- o BSSID, source address, destination address

Destination address is often of Broadcast type.

- o SSID of infrastructure network

- o AP/Router supported rates

Active Scanning

This involves transmission of broadcast probe request frames by the wireless client device and the AP/routers responding back with a directed (unicast) probe response frames (destination is the same client device that transmits the probe request frames). Likewise, a scan list result will be generated from the probe response frames received.

“Hidden” AP/router

In the case of a “hidden” AP/router, only active scan is used and the wireless client knows the SSID of the wireless network it desires to connect to. The wireless client will transmit a directed probe request frame to this “hidden” AP/router and the “hidden” AP/router will respond back with the corresponding probe response frames.

Refer to the figure below.

Once the wireless client knows the particular infrastructure network it wishes to join, the authentication and association process will be initiated.

Step 4 : Authentication

To initiate wireless connection to the AP/router, authentication process will have to first take place.

The wireless client device will transmit a directed authentication frame to the desired AP/router. The AP/Router will respond by transmitting a directed authentication to the same wireless client device to indicate the authentication status.

Within the authentication frame from the AP/router, it will contain

- o BSSID, source address, destination address

Destination address will be the wireless device which has transmitted the authentication (directed).

- o Authentication Algorithm

This contained the authentication algorithm used, such as Open System.

- o Status Code

Status Code will indicate whether authentication is a success or failure.

Step 5 : Association

When the authentication process is completed, this will be followed by the association process.

The wireless client device will transmit a directed association request frame to the desired AP/router. The AP/Router will respond by transmitting a directed association response frame to the same wireless device to indicate the association status.

Within the association response frame from the AP/router, it will contain

- o BSSID, source address, destination address

Destination address will be the wireless device which has transmitted the association request (directed).

- o Status Code and Association ID (AID)

Status Code will indicate whether association is a success or failure. If association is successful, an unique AID will be assigned.

Integration of WPS into 802.11 joining operation

Below lists the overall sequences

- Scanning, Authentication, Association
- WPS Frame Exchanges (EAP protocol)
- Deauthentication or Disassociation
 - o Some APs are found to transmit disassociation instead of deauthentication frame.

Provision needs to be made to handle receipt of disassociation frame.

- Authentication, Association
- EAPOL 4-way handshake or 802.1X-authentication

Disconnection from a wireless network

If the device is disconnected according to the 802.11 specifications, either of the following procedures will take place.

- Disassociation
- Deauthentication

Or

- Deauthentication

When an AP sends a deauthentication notice to an associated STA, the association shall also be terminated

Index

—
 _TFTP_ACCESS_ERROR 272
 _TFTP_ACCESS_ERROR enumeration 272
 _TFTP_FILE_MODE 272
 _TFTP_FILE_MODE enumeration 272
 _TFTP_RESULT 273
 _TFTP_RESULT enumeration 273
 _tftpError 276
 _tftpError variable 276
 _tftpSocket 276
 _tftpSocket variable 276

8

802.11 AP/Router Configuration Settings 329

A

accept 175
 accept function 175
 Access Point Compatibility 326
 Accessing the Demo Application 132
 activeUDPSocket 286
 activeUDPSocket variable 286
 Additional Features 164
 Address 160
 Ad-hoc Network 291
 Advanced MPFS2 Settings 122
 AF_INET 175
 AF_INET macro 175
 Announce 169
 Announce Stack Members 169
 AnnounceIP 169
 AnnounceIP function 169
 APP_CONFIG Structure 154
 ARCFOUR 170
 ARCFOUR Public Members 170
 ARCFOUR_CTX 170
 ARCFOUR_CTX structure 170
 ARP 170
 ARP Internal Members 173

ARP Public Members 171
 ARP Stack Members 172
 arp_app_callbacks 171
 arp_app_callbacks structure 171
 ARP_IP 173
 ARP_IP macro 173
 ARP_OPERATION_REQ 173
 ARP_OPERATION_REQ macro 173
 ARP_OPERATION_RESP 173
 ARP_OPERATION_RESP macro 173
 ARP_REQ 172
 ARP_REQ macro 172
 ARP_RESP 172
 ARP_RESP macro 172
 ARPInit 172
 ARPInit function 172
 ARPPProcess 172
 ARPPProcess function 172
 ARPRegisterCallbacks 171
 ARPRegisterCallbacks function 171
 Authentication 142
 Available Demos 135

B

Berkeley (BSD) Sockets 174
 BerkeleySocketInit 182
 BerkeleySocketInit function 182
 bind 175
 bind function 175
 BSD Sockets 168
 BSD Wrapper Internal Members 182
 BSD Wrapper Public Members 174
 BSD Wrapper Stack Members 182
 BSD_SCK_STATE 182
 BSD_SCK_STATE type 182
 BSDSocket 175
 BSDSocket structure 175
 Building MPFS2 Images 121

C

closesocket 176
 closesocket function 176

Configure your WiFi Access Point 130
Configuring the Stack 158
Configuring WiFi Security 132
connect 176
connect function 176
Connecting to the Network 131
Connection Algorithm Internal Members 300
Connection Algorithm Public Members 296
Connection Manager Public Members 300
Connection Profile Internal Members 296
Connection Profile Public Members 295
Cookies 144
Cooperative Multitasking 155
curHTTP 203
curHTTP variable 203
curHTTPID 209
curHTTPID variable 209

D

DDNS_CHECKIP_SERVER 190
DDNS_CHECKIP_SERVER macro 190
DDNS_DEFAULT_PORT 190
DDNS_DEFAULT_PORT macro 190
DDNS_POINTERS 185
DDNS_POINTERS structure 185
DDNS_SERVICES 186
DDNS_SERVICES enumeration 186
DDNS_STATUS 187
DDNS_STATUS enumeration 187
DDNSClient 188
DDNSClient variable 188
DDNSForceUpdate 188
DDNSForceUpdate function 188
DDNSGetLastIP 188
DDNSGetLastIP function 188
DDNSGetLastStatus 188
DDNSGetLastStatus function 188
DDNSInit 189
DDNSInit function 189
DDNSSetService 189
DDNSSetService function 189
DDNSTask 189

DDNSTask function 189
Demo Board Information 125
Demo Compatibility Table 134
Demo Information 134
DiscoveryTask 169
DiscoveryTask function 169
DNS Client 183
DNS Internal Members 184
DNS Public Members 183
DNS_TYPE_A 184
DNS_TYPE_A macro 184
DNS_TYPE_MX 184
DNS_TYPE_MX macro 184
DNSBeginUsage 183
DNSBeginUsage function 183
DNSEndUsage 183
DNSEndUsage function 183
DNSResolveROM 184
DNSResolveROM function 184
Dynamic DNS Client 184
Dynamic DNS Internal Members 189
Dynamic DNS Public Members 185
Dynamic DNS Stack Members 189
Dynamic Variables 142

E

E-mail (SMTP) Demo 145
Encounter issues after upgrading MRF24WB0M RF module
Firmware version 0x1207 340
Error Scenario And Possible Causes 345
Explorer 16 128
External Storage 158

F

Forms using GET 143
Forms using POST 144

G

GenerateRandomDWORD 193
GenerateRandomDWORD function 193
Generating Server Certificates 241
Generic TCP Client 146

Generic TCP Server 147
 gethostname 176
 gethostname function 176
 Getting Help 17
 Getting Started 125

H

Handling of maximum length SSID 335
 Hardware Configuration 158
 HASH_SUM 191
 HASH_SUM structure 191
 HASH_TYPE 192
 HASH_TYPE enumeration 192
 HashAddROMData 191
 HashAddROMData function 191
 Hashes 190
 Hashes Internal Members 192
 Hashes Public Members 190
 Hashes Stack Members 191
 Helpers 193
 Helpers Public Members 193
 Hibernate Mode 334
 How the Stack Works 153
 How to determine new IP address assigned 341
 How to fix MRF24WB0M / MRF24WG0M transmission rates 341
 How to increase TCP throughput 342
 HTTP_CACHE_LEN 209
 HTTP_CACHE_LEN macro 209
 HTTP_CONN 204
 HTTP_CONN structure 204
 HTTP_FILE_TYPE 210
 HTTP_FILE_TYPE enumeration 210
 HTTP_IO_RESULT 204
 HTTP_IO_RESULT enumeration 204
 HTTP_MAX_DATA_LEN 210
 HTTP_MAX_DATA_LEN macro 210
 HTTP_MIN_CALLBACK_FREE 210
 HTTP_MIN_CALLBACK_FREE macro 210
 HTTP_PORT 211
 HTTP_PORT macro 211
 HTTP_READ_STATUS 205
 HTTP_READ_STATUS enumeration 205
 HTTP_STATUS 211
 HTTP_STATUS enumeration 211
 HTTP_STUB 212
 HTTP_STUB structure 212
 HTTP_TIMEOUT 212
 HTTP_TIMEOUT macro 212
 HTTP2 Authentication 200
 HTTP2 Compression 202
 HTTP2 Cookies 202
 HTTP2 Dynamic Variables 195
 HTTP2 Features 195
 HTTP2 Form Processing 197
 HTTP2 Internal Members 208
 HTTP2 Public Members 203
 HTTP2 Server 194
 HTTP2 Stack Members 208
 HTTPExecuteGet 205
 HTTPExecuteGet function 205
 HTTPExecutePost 206
 HTTPExecutePost function 206
 HTTPGetROMArg 212
 HTTPGetROMArg function 212
 HTTPInit 208
 HTTPInit function 208
 HTTPReadPostPair 207
 HTTPReadPostPair macro 207
 HTTPS_PORT 212
 HTTPS_PORT macro 212
 HTTPServer 208
 HTTPServer function 208
 httpStubs 213
 httpStubs variable 213
 HW_ETHERNET 173
 HW_ETHERNET macro 173

I

ICMP 214
 ICMP Internal Members 215
 ICMP Public Members 214
 ICMPBeginUsage 214
 ICMPBeginUsage function 214
 ICMPEndUsage 215

ICMPEndUsage function 215
 ICMPGetReply 214
 ICMPGetReply function 214
 ICMPSendPingToHostROM 215
 ICMPSendPingToHostROM macro 215
 ifconfig Commands 138
 in_addr 176
 in_addr structure 176
 INADDR_ANY 177
 INADDR_ANY macro 177
 Infrastructure Network 291
 Initialization 154
 Initialization Structure 167
 Introduction 17
 INVALID_SOCKET 258
 INVALID_SOCKET macro 258
 INVALID_TCP_PORT 178
 INVALID_TCP_PORT macro 178
 INVALID_UDP_PORT 282
 INVALID_UDP_PORT macro 282
 INVALID_UDP_SOCKET 282
 INVALID_UDP_SOCKET macro 282
 IP Address 161
 IP_ADDR_ANY 178
 IP_ADDR_ANY macro 178
 iperf Example 140
 IPPROTO_IP 179
 IPPROTO_IP macro 179
 IPPROTO_TCP 179
 IPPROTO_TCP macro 179
 IPPROTO_UDP 179
 IPPROTO_UDP macro 179
 iwconfig Commands 136
 iwpriv Commands 139

L

leftRotateDWORD 193
 leftRotateDWORD function 193
 Legal Information 18
 Library Interface 169
 listen 179
 listen function 179

M

MAC Address 160
 Main File 154
 Main Loop 155
 Management Scan Message Conflict 334
 Maximum Scan Results 305
 MD5AddROMData 192
 MD5AddROMData function 192
 Memory Allocation 165
 Microchip TCP/IP Discoverer 123
 Missing DHCP Client Name 343
 MPFS_FAT_RECORD 222
 MPFS_FAT_RECORD structure 222
 MPFS_HANDLE 217
 MPFS_HANDLE type 217
 MPFS_INVALID 217
 MPFS_INVALID macro 217
 MPFS_INVALID_FAT 223
 MPFS_INVALID_FAT macro 223
 MPFS_INVALID_HANDLE 217
 MPFS_INVALID_HANDLE macro 217
 MPFS_PTR 221
 MPFS_PTR type 221
 MPFS_SEEK_MODE 217
 MPFS_SEEK_MODE enumeration 217
 MPFS_STUB 221
 MPFS_STUB structure 221
 MPFS_WRITE_PAGE_SIZE 221
 MPFS_WRITE_PAGE_SIZE macro 221
 MPFS2 121, 215
 MPFS2 Command Line Options 123
 MPFS2 Internal Members 220
 MPFS2 Public Members 216
 MPFS2 Stack Members 220
 MPFS2_FLAG_HASINDEX 221
 MPFS2_FLAG_HASINDEX macro 221
 MPFS2_FLAG_ISZIPPED 222
 MPFS2_FLAG_ISZIPPED macro 222
 MPFSClose 218
 MPFSClose function 218
 MPFSFormat 218

MPFSFormat function 218
 MPFSGetBytesRem 218
 MPFSGetBytesRem function 218
 MPFSGetEndAddr 218
 MPFSGetEndAddr function 218
 MPFSGetFlags 218
 MPFSGetFlags function 218
 MPFSGetID 219
 MPFSGetID function 219
 MPFSGetMicrotime 219
 MPFSGetMicrotime function 219
 MPFSGetPosition 219
 MPFSGetPosition function 219
 MPFSGetSize 219
 MPFSGetSize function 219
 MPFSGetStartAddr 219
 MPFSGetStartAddr function 219
 MPFSGetTimestamp 220
 MPFSGetTimestamp function 220
 MPFSInit 220
 MPFSInit function 220
 MPFSOpenROM 222
 MPFSOpenROM function 222
 MPFSTell 222
 MPFSTell macro 222
 MRF24WB/MRF24WG PICTail Daughter Board 127
 MRF24WB0M advertised supported rates of 1, 2, 5.5 and 11 Mbps 339
 MRF24WB0M assert failures whe using <iwconfig scan> command 337
 MRF24WB0M Compatibility with AP/Routers 340
 Multicast Filters : Hardware vs Software 337

N

NBNS 223
 NBNS Stack Members 223
 NBNSTask 223
 NBNSTask function 223
 Network Switch or Change 333
 Null String ESSID 330

P

Performance Test Internal Members 224

Performance Test Stack Members 224
 Performance Tests 224
 Peripheral Usage 125
 PIC18 Explorer Development Board 128
 PIC18F87J11 Config 160
 PIC18F87J11 Plug-In-Module (PIM) 126
 PIC24FJ256GB110 Plug-In-Module (PIM) 125
 PIC32MX795F512L Plug-In-Module (PIM) 126
 PIC32MX7XX Config 159
 Ping (ICMP) Demo 147
 Power Save Internal Members 314
 Power Save Public Members 313
 Programming and First Run 129
 Protocol Configuration 162
 Protocol Macros and Files 163

R

Read back RF module Firmware version 330
 Reboot 232
 Reboot Stack Members 233
 RebootTask 233
 RebootTask function 233
 recv 177
 recv function 177
 recvfrom 178
 recvfrom function 178
 Release Notes 19
 Required Files 153
 RESERVED_HTTP_MEMORY 213
 RESERVED_HTTP_MEMORY macro 213
 RESERVED_SSL_MEMORY 247
 RESERVED_SSL_MEMORY macro 247
 RF Module Firmware Update 331
 ROMStringToIPAddress 194
 ROMStringToIPAddress function 194
 RSA 233
 RSA Internal Members 236
 RSA Public Members 233
 RSA Stack Members 235
 RSA_DATA_FORMAT 234
 RSA_DATA_FORMAT enumeration 234
 RSA_KEY_WORDS 237

- RSA_KEY_WORDS macro 237
- RSA_OP 234
- RSA_OP enumeration 234
- RSA_PRIME_WORDS 237
- RSA_PRIME_WORDS macro 237
- RSA_STATUS 235
- RSA_STATUS enumeration 235
- RSABeginDecrypt 235
- RSABeginDecrypt macro 235
- RSABeginEncrypt 236
- RSABeginEncrypt macro 236
- RSAEndDecrypt 236
- RSAEndDecrypt macro 236
- RSAEndEncrypt 236
- RSAEndEncrypt macro 236
- RSAEndUsage 234
- RSAEndUsage function 234
- RSAINit 236
- RSAINit function 236
- RSAStep 234
- RSAStep function 234

- S**
- Scan Operation and Scan Results 303
- Scan Public Members 305
- send 178
- send function 178
- sendto 178
- sendto function 178
- SHA1AddROMData 192
- SHA1AddROMData function 192
- Shorter Scan or Connection Duration 304
- sktHTTP 208
- sktHTTP macro 208
- SM_HTTP2 213
- SM_HTTP2 enumeration 213
- SM_RSA 237
- SM_RSA enumeration 237
- SM_SSL_RX_SERVER_HELLO 247
- SM_SSL_RX_SERVER_HELLO enumeration 247
- SMTP Client 225
- SMTP Client Examples 225
- SMTP Client Internal Members 232
- SMTP Client Long Message Example 226
- SMTP Client Public Members 227
- SMTP Client Short Message Example 225
- SMTP Client Stack Members 231
- SMTP_CONNECT_ERROR 227
- SMTP_CONNECT_ERROR macro 227
- SMTP_POINTERS 228
- SMTP_POINTERS structure 228
- SMTP_RESOLVE_ERROR 229
- SMTP_RESOLVE_ERROR macro 229
- SMTP_SUCCESS 229
- SMTP_SUCCESS macro 229
- SMTPBeginUsage 230
- SMTPBeginUsage function 230
- SMTPClient 230
- SMTPClient variable 230
- SMTPEndUsage 230
- SMTPEndUsage function 230
- SMTPFlush 230
- SMTPFlush function 230
- SMTPIsBusy 230
- SMTPIsBusy function 230
- SMTPIsPutReady 231
- SMTPIsPutReady function 231
- SMTPPutDone 231
- SMTPPutDone function 231
- SMTPPutROMArray 232
- SMTPPutROMArray function 232
- SMTPPutROMString 232
- SMTPPutROMString function 232
- SMTPSendMail 231
- SMTPSendMail function 231
- SMTPTask 231
- SMTPTask function 231
- SNTP Client 238
- SNTP Client Internal Members 239
- SNTP Client Public Members 238
- SNTP Client Stack Members 238
- SNTPClient 239
- SNTPClient function 239
- SNTPGetUTCSeconds 238

SNTPGetUTCSeconds function 238	SSL_BUFFER_SPACE 249
SOCK_DGRAM 179	SSL_BUFFER_SPACE macro 249
SOCK_DGRAM macro 179	SSL_CHANGE_CIPHER_SPEC 249
SOCK_STREAM 180	SSL_CHANGE_CIPHER_SPEC macro 249
SOCK_STREAM macro 180	SSL_HANDSHAKE 249
sockaddr 180	SSL_HANDSHAKE macro 249
SOCKADDR 180	SSL_HASH_SIZE 250
sockaddr structure 180	SSL_HASH_SIZE macro 250
SOCKADDR type 180	SSL_HASH_SPACE 250
sockaddr_in 180	SSL_HASH_SPACE macro 250
SOCKADDR_IN 181	SSL_INVALID_ID 243
sockaddr_in structure 180	SSL_INVALID_ID macro 243
SOCKADDR_IN type 181	SSL_KEYS 250
socket 181	SSL_KEYS structure 250
SOCKET 181	SSL_KEYS_SIZE 251
socket function 181	SSL_KEYS_SIZE macro 251
SOCKET type 181	SSL_KEYS_SPACE 251
Socket Types 166	SSL_KEYS_SPACE macro 251
SOCKET_CNXN_IN_PROGRESS 181	SSL_MESSAGES 251
SOCKET_CNXN_IN_PROGRESS macro 181	SSL_MESSAGES enumeration 251
SOCKET_DISCONNECTED 181	SSL_MIN_SESSION_LIFETIME 246
SOCKET_DISCONNECTED macro 181	SSL_MIN_SESSION_LIFETIME macro 246
SOCKET_ERROR 182	SSL_PKEY_INFO 243
SOCKET_ERROR macro 182	SSL_PKEY_INFO structure 243
SOCKET_INFO 265	SSL_RSA_CLIENT_SIZE 244
SOCKET_INFO type 265	SSL_RSA_CLIENT_SIZE macro 244
Sockets 165	SSL_RSA_LIFETIME_EXTENSION 246
SoftAP Network 291	SSL_RSA_LIFETIME_EXTENSION macro 246
SSL 239	SSL_SESSION 252
SSL Internal Members 246	SSL_SESSION structure 252
SSL Public Members 242	SSL_SESSION_SIZE 252
SSL Stack Members 244	SSL_SESSION_SIZE macro 252
SSL_ALERT 248	SSL_SESSION_SPACE 253
SSL_ALERT macro 248	SSL_SESSION_SPACE macro 253
SSL_ALERT_LEVEL 248	SSL_SESSION_STUB 253
SSL_ALERT_LEVEL enumeration 248	SSL_SESSION_STUB structure 253
SSL_APPLICATION 248	SSL_SESSION_TYPE 253
SSL_APPLICATION macro 248	SSL_SESSION_TYPE enumeration 253
SSL_BUFFER 248	SSL_STATE 244
SSL_BUFFER union 248	SSL_STATE type 244
SSL_BUFFER_SIZE 249	SSL_STUB 254
SSL_BUFFER_SIZE macro 249	SSL_STUB structure 254

SSL_STUB_SIZE 255	TCP_ADJUST_PRESERVE_TX macro 259
SSL_STUB_SIZE macro 255	TCP_OPEN_IP_ADDRESS 259
SSL_STUB_SPACE 255	TCP_OPEN_IP_ADDRESS macro 259
SSL_STUB_SPACE macro 255	TCP_OPEN_NODE_INFO 259
SSL_SUPPLEMENTARY_DATA_TYPES 243	TCP_OPEN_NODE_INFO macro 259
SSL_SUPPLEMENTARY_DATA_TYPES enumeration 243	TCP_OPEN_RAM_HOST 260
SSL_VERSION 255	TCP_OPEN_RAM_HOST macro 260
SSL_VERSION macro 255	TCP_OPEN_ROM_HOST 260
SSL_VERSION_HI 255	TCP_OPEN_ROM_HOST macro 260
SSL_VERSION_HI macro 255	TCP_OPEN_SERVER 260
SSL_VERSION_LO 256	TCP_OPEN_SERVER macro 260
SSL_VERSION_LO macro 256	TCP_SOCKET 266
SSLFinishPartialRecord 256	TCP_SOCKET type 266
SSLFinishPartialRecord macro 256	TCP_STATE 266
SSLFlushPartialRecord 256	TCP_STATE type 266
SSLFlushPartialRecord macro 256	TCPClose 261
SSLInit 245	TCPClose function 261
SSLInit function 245	TCPConnect 260
SSLRxHandshake 256	TCPConnect macro 260
SSLRxHandshake function 256	TCPDiscard 261
Stack API 169	TCPDiscard function 261
Stack Architecture 153	TCPDisconnect 261
Standalone Commands 135	TCPDisconnect function 261
strnchr 194	TCPFind 261
strnchr function 194	TCPFind macro 261
	TCPFindArray 261
	TCPFindArray macro 261
	TCPFindROMArray 262
	TCPFindROMArray macro 262
	TCPFindROMArrayEx 262
	TCPFindROMArrayEx macro 262
	TCPFlush 262
	TCPFlush function 262
	TCPGetRemoteInfo 262
	TCPGetRemoteInfo function 262
	TCPGetRxFIFOFree 262
	TCPGetRxFIFOFree function 262
	TCPGetRxFIFOFull 263
	TCPGetRxFIFOFull macro 263
	TCPGetTxFIFOFree 263
	TCPGetTxFIFOFree macro 263
	TCPGetTxFIFOFull 263
TCB 265	
TCB type 265	
TCB_STUB 265	
TCB_STUB type 265	
TCP 256	
TCP Internal Members 266	
TCP Public Members 257	
TCP Stack Members 265	
TCP_ADJUST_GIVE_REST_TO_RX 258	
TCP_ADJUST_GIVE_REST_TO_RX macro 258	
TCP_ADJUST_GIVE_REST_TO_TX 259	
TCP_ADJUST_GIVE_REST_TO_TX macro 259	
TCP_ADJUST_PRESERVE_RX 259	
TCP_ADJUST_PRESERVE_RX macro 259	
TCP_ADJUST_PRESERVE_TX 259	

T

TCPGetTxFIFOFull function 263	Telnet Stack Members 267
TCPInit 266	TelnetTask 267
TCPInit function 266	TelnetTask function 267
TCPIP Library 16	TFTP 268
TCPIP WiFi Console Demo App 135	TFTP Internal Members 276
TCPIP WiFi Demo App 141	TFTP Public Members 268
TCPIP WiFi Demo Modules 141	TFTP Stack Members 275
TCPIP WiFi EasyConfig Demo 150	TFTP_ACCESS_ERROR 272
TCPIP WiFi G Demo 135	TFTP_ACCESS_ERROR enumeration 272
TCPIsConnected 263	TFTP_ARP_TIMEOUT_VAL 275
TCPIsConnected function 263	TFTP_ARP_TIMEOUT_VAL macro 275
TCPIsGetReady 263	TFTP_CHUNK_DESCRIPTOR 273
TCPIsGetReady function 263	TFTP_CHUNK_DESCRIPTOR type 273
TCPIsPutReady 264	TFTP_END_OF_FILE enumeration member 273
TCPIsPutReady function 264	TFTP_ERROR enumeration member 273
TCPIsSSL 243	TFTP_ERROR_ACCESS_VIOLATION enumeration member 272
TCPIsSSL function 243	TFTP_ERROR_DISK_FULL enumeration member 272
TCPListen 264	TFTP_ERROR_FILE_EXISTS enumeration member 272
TCPListen macro 264	TFTP_ERROR_FILE_NOT_FOUND enumeration member 272
TCPPerformanceTask 224	TFTP_ERROR_INVALID_OPERATION enumeration member 272
TCPPerformanceTask function 224	TFTP_ERROR_NO_SUCH_USE enumeration member 272
TCPPutROMArray 264	TFTP_ERROR_NOT_DEFINED enumeration member 272
TCPPutROMArray macro 264	TFTP_ERROR_UNKNOWN_TID enumeration member 272
TCPPutROMString 264	TFTP_FILE_MODE 272
TCPPutROMString macro 264	TFTP_FILE_MODE enumeration 272
TCPSSLGetPendingTxSize 245	TFTP_FILE_MODE_READ enumeration member 272
TCPSSLGetPendingTxSize function 245	TFTP_FILE_MODE_WRITE enumeration member 272
TCPSSLHandleIncoming 245	TFTP_GET_TIMEOUT_VAL 276
TCPSSLHandleIncoming function 245	TFTP_GET_TIMEOUT_VAL macro 276
TCPSSLHandshakeComplete 245	TFTP_MAX_RETRIES 276
TCPSSLHandshakeComplete function 245	TFTP_MAX_RETRIES macro 276
TCPSSLIsHandshaking 243	TFTP_NOT_READY enumeration member 273
TCPSSLIsHandshaking function 243	TFTP_OK enumeration member 273
TCPStartSSLServer 245	TFTP_RESULT 273
TCPStartSSLServer function 245	TFTP_RESULT enumeration 273
TCPTick 266	TFTP_RETRY enumeration member 273
TCPTick function 266	TFTP_TIMEOUT enumeration member 273
TCPWasReset 264	TFTP_UPLOAD_COMPLETE 273
TCPWasReset function 264	TFTP_UPLOAD_COMPLETE macro 273
Telnet 267	TFTP_UPLOAD_CONNECT 273
Telnet Internal Members 267	
Telnet Public Members 267	

TFTP_UPLOAD_CONNECT macro 273	TFTPOpenROMFile macro 272
TFTP_UPLOAD_CONNECT_TIMEOUT 274	TICK 278
TFTP_UPLOAD_CONNECT_TIMEOUT macro 274	Tick Internal Members 280
TFTP_UPLOAD_GET_DNS 274	Tick Module 277
TFTP_UPLOAD_GET_DNS macro 274	Tick Public Members 277
TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT 274	Tick Stack Functions 279
TFTP_UPLOAD_HOST_RESOLVE_TIMEOUT macro 274	TICK variable 278
TFTP_UPLOAD_RESOLVE_HOST 274	TICK_HOUR 278
TFTP_UPLOAD_RESOLVE_HOST macro 274	TICK_HOUR macro 278
TFTP_UPLOAD_SEND_DATA 274	TICK_MINUTE 278
TFTP_UPLOAD_SEND_DATA macro 274	TICK_MINUTE macro 278
TFTP_UPLOAD_SEND_FILENAME 275	TICK_SECOND 278
TFTP_UPLOAD_SEND_FILENAME macro 275	TICK_SECOND macro 278
TFTP_UPLOAD_SERVER_ERROR 275	TickGet 279
TFTP_UPLOAD_SERVER_ERROR macro 275	TickGet function 279
TFTP_UPLOAD_WAIT_FOR_CLOSURE 275	TickGetDiv256 279
TFTP_UPLOAD_WAIT_FOR_CLOSURE macro 275	TickGetDiv256 function 279
TFTPClose 269	TickGetDiv64K 279
TFTPClose macro 269	TickGetDiv64K function 279
TFTPCloseFile 269	TickInit 279
TFTPCloseFile function 269	TickInit function 279
TFTPGet 270	TICKS_PER_SECOND 280
TFTPGet function 270	TICKS_PER_SECOND macro 280
TFTPGetError 270	TickUpdate 280
TFTPGetError macro 270	TickUpdate function 280
TFTPGetUploadStatus 273	tWFDeviceInfoStruct 326
TFTPGetUploadStatus function 273	tWFDeviceInfoStruct structure 326
TFTPIsFileClosed 270	tWFScanResult 306
TFTPIsFileClosed function 270	tWFScanResult structure 306
TFTPIsFileOpened 270	tWFWpsCred 311
TFTPIsFileOpened function 270	tWFWpsCred structure 311
TFTPIsFileOpenReady 271	Tx Power Control Public Members 312
TFTPIsFileOpenReady macro 271	
TFTPIsGetReady 271	
TFTPIsGetReady function 271	
TFTPIsOpened 271	
TFTPIsOpened function 271	
TFTPIsPutReady 271	
TFTPIsPutReady function 271	
TFTPOpen 272	
TFTPOpen function 272	
TFTPOpenROMFile 272	

U

UART-to-TCP Bridge 148
 UDP 280
 UDP Internal Members 286
 UDP Public Members 281
 UDP Sockets 167
 UDP Stack Members 285
 UDP_HEADER 286
 UDP_HEADER type 286

UDP_OPEN_IP_ADDRESS 284
 UDP_OPEN_IP_ADDRESS macro 284
 UDP_OPEN_NODE_INFO 284
 UDP_OPEN_NODE_INFO macro 284
 UDP_OPEN_RAM_HOST 285
 UDP_OPEN_RAM_HOST macro 285
 UDP_OPEN_ROM_HOST 285
 UDP_OPEN_ROM_HOST macro 285
 UDP_OPEN_SERVER 285
 UDP_OPEN_SERVER macro 285
 UDP_PORT 287
 UDP_PORT type 287
 UDP_SOCKET 282
 UDP_SOCKET type 282
 UDP_SOCKET_INFO 287
 UDP_SOCKET_INFO type 287
 UDPClose 283
 UDPClose function 283
 UDPDiscard 283
 UDPDiscard function 283
 UDPFlush 283
 UDPFlush function 283
 UDPInit 285
 UDPInit function 285
 UDPisGetReady 284
 UDPisGetReady function 284
 UDPisOpened 284
 UDPisOpened function 284
 UDPisPutReady 284
 UDPisPutReady function 284
 UDPOpen 282
 UDPOpen macro 282
 UDPPerformanceTask 224
 UDPPerformanceTask function 224
 UDPRxCount 287
 UDPRxCount variable 287
 UDPSocketInfo 287
 UDPSocketInfo variable 287
 UDPTask 286
 UDPTask function 286
 UDPTxCount 287
 UDPTxCount variable 287

ultoa 194
 ultoa macro 194
 UNKNOWN_SOCKET 258
 UNKNOWN_SOCKET macro 258
 Uploading Pre-built MPFS2 Images 122
 Uploading Web Pages 131
 Use of macro #define MY_DEFAULT_CHANNEL_LIST 305
 Using the Stack 153
 Utilities 121

W

Web Page Demos 141
 WF_CAGetElements 297
 WF_CAGetElements function 297
 WF_CASetElements 297
 WF_CASetElements function 297
 WF_CMDDisconnect 301
 WF_CMDDisconnect function 301
 WF_CMGetConnectContext 301
 WF_CMGetConnectContext function 301
 WF_CMInfoGetFSMStats 301
 WF_CMInfoGetFSMStats function 301
 WF_DisableModuleConnectionManager 302
 WF_DisableModuleConnectionManager function 302
 WF_EnableSWMultiCastFilter 323
 WF_EnableSWMultiCastFilter function 323
 WF_GetDeviceInfo 322
 WF_GetDeviceInfo function 322
 WF_GetMacStats 323
 WF_GetMacStats function 323
 WF_HibernateEnable 313
 WF_HibernateEnable function 313
 WF_MulticastSetConfig 324
 WF_MulticastSetConfig function 324
 WF_PsPollDisable 314
 WF_PsPollDisable function 314
 WFCAElementsStruct 298
 WFCAElementsStruct structure 298
 WFCPElementsStruct 295
 WFCPElementsStruct structure 295
 WFHibernate 314
 WFHibernate structure 314

- WFMacStatsStruct 325
- WFMacStatsStruct structure 325
- WFMulticastConfigStruct 326
- WFMulticastConfigStruct structure 326
- Wi-Fi API 288
- Wi-Fi Connection Algorithm 296
- Wi-Fi Connection Manager 300
- Wi-Fi Connection Profile 294
- Wi-Fi Demo Board Hardware Setup 127
- Wi-Fi Direct Network 294
- Wi-Fi G Demo Board 129
- Wi-Fi Miscellaneous 321
- Wi-Fi Miscellaneous Public Members 322
- WiFi MRF24WG Events 317
- Wi-Fi Network Topologies 290
- Wi-Fi Power Save 312
- Wi-Fi Process Event 314
- Wi-Fi Protected Access (WPA/WPA2) 308
- Wi-Fi Protected Setup (WPS) 308
- Wi-Fi Protected Setup (WPS) Issues 332
- Wi-Fi Scan 302
- Wi-Fi Security 306
- WiFi Troubleshooting Tips 330
- Wi-Fi Tx Power Control 312
- Wired Equivalent Privacy (WEP) 307
- Wireless Packets Analysis 347

Z

- Zero Configuration (ZeroConf) 149